САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИСТЕТ

Кафедра вычислительной физики

О.Л. Стесик, М.М. Степанова

ОСНОВЫ ПРОГРАММИРОВАНИЯ
КУРС НА БАЗЕ ЯЗЫКА ПРОГРАММИРОВАНИЯ JAVA

Учебно-методическое пособие

Санкт-Петербург
2016 г.

Рецензенты:

доцент кафедры вычислительной физики,
кандидат физ.-мат. наук, Е.А. Яревский
доцент кафедры ядерно-физических методов исследования,
кандидат физ.-мат. наук , А.В. Комолкин

Печатается по решению Ученого совета физического факультета СПбГУ.

О.Л. Стесик, М.М. Степанова
Основы программирования. Курс на базе языка программирования Java.
СПб., 2016. 62 с.

Учебно-методическое пособие содержит начальные сведения о языках и базовых идеях программирования, иллюстрируемых примерами на языке программирования Java. Материал предназначен для изучения во втором семестре студентами первого курса Основной образовательной программы «Физика» по направлению 03.03.02 «Физика» в рамках учебной дисциплины «Практикум по информатике».

ОГЛАВЛЕНИЕ

Предисловие

Основная цель курса «Практикум по информатике» состоит в изучении методов алгоритмизации и языков программирования, подготовке студента к практическому курсу по численным методам, создании основы для выполнения заданий по компьютерному моделированию и других смежных дисциплин.

Задачи учебных занятий: на примере выбранного языка программирования развить и закрепить навыки решения проблем путем разработки программ, а также выделить в процессе изучения аспекты, присущие многим языкам программирования. В процессе занятий изучаются основные понятия и подходы программирования: цикл разработки программ, средства разработки; ключевые слова, переменные и типы переменных, встроенные операции, выражения, операторы описания и присваивания; циклические и управляющие конструкции; ввод-вывод данных; структура программы, локальные и глобальные переменные, описания и области видимости переменных; функции и подпрограммы, формальные и фактические параметры; одномерные и многомерные массивы; исполняемые и неисполняемые программы, библиотеки.

В той версии курса, для которой разработано данное учебно-методическое пособие, основной акцент сделан на развитии умения быстро изучить язык программирования, опираясь на общие принципы и понятия. В изложении материала выделяются необходимые в изучении любого языка программирования вопросы и не подчеркиваются детали и тонкости.

Курс поддерживается информационным сайтом (URL http://gr2.phys.spbu.ru/Informatika), где размещены базовые и текущие учебные материалы.

Курс разделен на 9 относительно независимых этапов, без которых не может обойтись изучение ни одного языка программирования. Прохождение каждого этапа иллюстрируется примером его выполнения с языком программирования Java. Этот язык программирования был выбран в качестве образца как относительно простой и распространенный, в котором реализованы и легко постигаются многие концептуальные понятия современного программирования. Следует подчеркнуть, что обучение языку программирования Java целью настоящего курса не является. Студентам предлагается самостоятельно выбрать язык программирования для изучения, и ознакомиться с ним, следуя схеме, изложенной в настоящем учебно-методическом пособии.

Тем же, кто не готов к самостоятельному изучению нового языка, остается курс базового языка с дополнениями, отчасти выходящими за рамки настоящего пособия. Студентам, изучающим Java, необходимо восполнять недостающий материал самостоятельно или, в сложных случаях, обращаться к дополнительному ма-

териалу. Отсылки к методическому пособию-дополнению для изучения Java обозначено пометкой «**Следующий уровень**».

Предупреждение

Курс проходит во втором семестре первого года обучения, после сдачи экзамена по предмету «Введение в информатику». Исходя из этого, предполагается владение понятиями «переменная», «оперативная память», «центральное процессорное устройство», «операционная система». Очень желательно владение понятием «подпрограмма» и понимание, зачем подпрограммы нужны.

1 Общая характеристика языка программирования

1.1 Пояснение к теме

Характеристика языка программирования – определение его принадлежности к той или иной категории языков программирования - позволяет заранее составить себе представление о предмете изучения.

Единой системы классификации пока еще не существует, но есть категории, не вызывающие сомнений. К ним относятся уровень: «высокий – низкий», и способ реализации: «компилируемый – интерпретируемый». Категория «парадигма» принята, но не несет определенности, а скорее является дополнительной характеристикой, так как свойства, определяемые ею, не являются взаимоисключающими. Добавим здесь категорию «характер», в которую включим две альтернативы «императивный – декларативный» и «процедурный – непроцедурный». (Эти качества иногда относят к парадигме).

Уровень языка определяется кругом понятий, которыми он оперирует. Для низкоуровневых языков – это понятия, используемые на уровне архитектуры системы команд : регистры, адреса оперативной памяти, данные типов «бит», «байт», «слово» и команды процессора. Понятия языков высокого уровня ближе понятиям человека-программиста: переменные, операторы, структуры данных, и команды, за каждой из которых скрывается множество рутинных действий низкого уровня.

Способ реализации определяется порядком превращения инструкций в код, исполняемый компьютером. Компилируемые языки привлекают программу-переводчик (компилятор) для создания исполнимого модуля с машинным кодом исходной программы. Интерпретируемые языки привлекают программу-интерпретатор команд и передают инструкции компьютеру через её посредство.

Парадигма языка программирования характеризует стиль написания программ на нем. Она определяется совокупностью базовых идей и понятий языка, и несет информацию о приоритетности целей применения языка. Значения в этой категории не исключают друг друга, хотя бы по той причине, что в них нет оконча-

тельной определенности. Однако определение парадигмы языка содержит полезную информацию о нем.

Под **характером** языка программирования (такой категории пока вообще нет) предлагается понимать разделение по признакам «процедурный – непроцедурный» и «императивный – декларативный». Процедурность предполагает разбиение программы на несколько более мелких частей (подпрограмм-процедур) и оформление всей программы в виде последовательных обращений к ним. Этот подход подразумевает использование одной программы с разными наборами данных. Непроцедурные языки программирования исходят из предположения поступления данных в виде потока, в котором могут оказаться фрагменты, нужные для обработки. В непроцедурных языках программа формируется как набор действий для разных шаблонов данных, в них нет самого понятия «последовательность действий». Императивность языка означает, что его операторы представляют собой указания типа «сделай вот это». В декларативных языках основным способом действий является декларация – объявление свойств или функций.

Ни один учебник ни по одному языку программирования не содержит сведений о его принадлежности к той или иной категории в явной форме. Но эти сведения легко найти в Интернете, так как дискуссий о классификации языков программирования и принадлежности того или иного языка к той или иной категории там достаточно. Классификация языка программирования позволит понять, каким образом компьютер исполняет программы, написанные на выбранном языке программирования, определить средства разработки программ и составить представление о концепции и стиле программирования на избранном языке.

1.2 Характеристика языка программирования Java

Выясним, как классифицируется язык программирования по указанным категориям.

по уровню:

это язык программирования высокого уровня. То есть,

- ✓ его представления данных близки к представлениям человека. (переменные, числа, символы), а не компьютера (адреса, регистры, машинные слова);
- ✓ инструкции (операторы) определяют законченное действие с точки зрения человека (вычислить выражение и сохранить полученное значение в переменной), а не в смысле единичной операции процессора (скопировать значение регистра в область памяти с указанным адресом);

по способу реализации:

компилируемый в интерпретируемый байт-код, то есть:

- ✓ исходная программа компилируется в универсальный, не зависящий от архитектуры и операционной системы, код (байт-код),
- ✓ интерпретатор байт-кода, а не операционная система, исполняет код, который получился в результате компиляции.

по парадигме:

строго объектно-ориентированный, то есть:

- ✓ этот язык приспособлен для создания новых композитных типов данных – классов;
- ✓ классы не являются программами;
- ✓ все подпрограммы оформляются в виде методов – членов класса;
- ✓ программирование на Java имеет два направления: программирование как создание новых типов и программирование как разработка исполняемых программ;
- ✓ при создании программ можно (и нужно) использовать наследование, инкапсуляцию и полиморфизм.
- ✓ исполняемая программа сводится к созданию экземпляров подходящих классов и вызовов процедур в виде обращения к методам экземпляров (или самих классов);
- ✓ нет никаких исполняемых программных единиц, кроме классов;

по характеру:

императивный, то есть,

- ✓ программа и подпрограммы-методы в Java являются последовательностью указаний, что надо сделать (компьютеру);

и, все-таки, процедурный, то есть

- ✓ вызов метода экземпляра класса представляет собой обращение к процедуре, а исполняемая программа считается главной (main) процедурой.

Как программировать на Java?

1. Исполняемую часть нужно представлять в виде одной или нескольких последовательностей обращений к процедурам как вызовам методов экземпляров или классов.
2. Для выбора процедур искать библиотечные классы, имеющие нужные методы.
3. Если нет подходящих классов, разработать новые, соответствующие задаче, классы.
4. Вернуться к пункту 1.

Какое программное обеспечение потребуется?

- ✓ текстовый редактор,
- ✓ компилятор,
- ✓ интерпретатор байт-кода.

Компилятор и интерпретатор байт-кода поставляются фирмой Oracle в составе комплекса Java Development Kit (инструментарий разработчика Java). Адрес страницы загрузки
http://www.oracle.com/technetwork/java/javase/downloads/index.html

Там же можно получить документацию к комплексу и найти ссылки на другую полезную информацию.

Вместе с JDK можно поставить программу для интегрированной разработки IDE NetBeans.

Для работы в классе:

JDK уже должен быть установлен, компилятор javac и интерпретатор java должны вызываться по своим коротким именам.

Текстовый редактор:

В графическом окружении gedit - стандартный системный текстовый редактор с графическим интерфейсом;

В командной строке nano – простой текстовый редактор.

Замечания, которые нужно учесть до перехода к примерам:

- ✓ файлы с текстами программ, должны иметь расширение java, иначе компилятор откажется их компилировать;
- ✓ все исполняемые java-программы являются классами, имена которых нужно придумывать самим. Возьмите за правило давать классам, которые вы создаете, и файлам, в которые вы их помещаете, одинаковые имена всегда, когда это возможно;
- ✓ регистр имеет значение: "class My", "Class my" и "clasS MY" - СоВеРшЕнНо разные строки;
- ✓ для работы с Java-программами создайте отдельный каталог подходящим именем (например, JavaPrograms) и размещайте все примеры и упражнения в нем.

1.2.1 Примеры программирования на языке Java

Исполняемая программа HelloWorld.java

В каталоге для работы с программами с помощью текстового редактора создадим файл HelloWorld.java следующего содержания:

```
class HelloWorld {
 public static void main(String[] args){
```

```
        System.out.println("Hello, world!");
    }
}
```

В командной строке вызовем компилятор:

```
$ javac HelloWorld.java
```

Если не появилось никаких сообщений, то компиляция выполнилась успешно. Убедимся в этом, проверив список файлов в каталоге:

```
$ ls
HelloWorld.class HelloWorld.java
```

Файл `Hello.class` был создан компилятором, он содержит байт-код, выполняемый интерпретатором (программа `java`):

```
$ java HelloWorld
Hello, world!
```

Обратите внимание на вызов интерпретатора: ему передается только имя класса, файл класса он разыскивает самостоятельно.

Естественно, что многое, если не всё, в этом примере непонятно. Рассмотрим каждую строку текста HelloWorld.java:

Первая строка – заголовок класса:

```
class HelloWorld {
```

Фигурная скобка открывает блок – тело класса.

Следующая инструкция - заголовок метода main.

Метод main является основной процедурой, только он может быть начальным адресом (точкой входа) программы.

```
public static void main(String[] args)
```

Атрибуты входного метода main обязательны:

`public` – устанавливает «общий» уровень доступа;

`static` – указывает на принадлежность метода классу;

`void` – метод ничего не возвращает, то есть, является процедурой, а не функцией;

Формальные параметры метода main также обязательны. Методу main передается массив строк, введенных в командной строке при вызове интерпретатора.

Квадратные скобки `[]` обозначают массив, `String` – название библиотечного класса, представляющего символьные строки, `args` – произвольное название локальной переменной для обращения к параметру. Фигурная скобка

```
{
```

открывает блок инструкций тела метода.

Инструкция

```
System.out.println("Hello, world!");
```

является вызовом метода `println(String s)` поля `out` класса `System` с аргументом `"Hello, world!"`. Первая закрывающая фигурная скобка

```
}
```

закрывается блок инструкций тела метода, вторая закрывающая фигурная скобка

```
}
```

закрывает блок тела класса.

Объектно-ориентированная версия примера: Hello.java

Программа HelloWorld.java не является примером объектно-ориентированного программирования, так как класс HelloWorld не является определением нового типа данных. По правилам Java единственной исполняемой программой может быть только класс. Это сделано для единообразия программных единиц. По сути же, класс HelloWorld является *исполняемой процедурой* – программой.

Определение понятия «класс».

Класс является определением производного типа данных, который представляет собой композицию из переменных и методов, связанных с этими переменными. Как правило, связь компонентов класса имеет смысловой характер, и сам класс является аналогом (абстрактным в той или иной степени) понятия реального мира. Хороший пример – строка символов (класс java.lang.String) или целое число (класс java.lang.Integer).

Класс представляет собой схему для построения по ней реальных структур-копий, заполненных конкретными значениями данных. Структуры-копии создаются во время выполнения программ и могут существовать, пока программа работает. Они называются экземплярами класса или объектами.

«Классовая» версия HelloWorld

Объектно-ориентированную версию этой же программы представляют классы `Hello` и `UseHello`. Класс `Hello` является полноценным определением нового типа данных; класс `UseHello`, подобно классу `HelloWorld`, представляет исполняемую программу, но эта программа построена на использовании типа `Hello`.

Класс `Hello` (из файла Hello.java)

```
class Hello{
```

```
String greeting="Hello";
String name="World";

String getHello(){
  return (greeting+", " + name +"!");
}
}
```

уже представляет собой группу переменных (полей) с одним методом, их использующим. Он не является исполняемой программой, так как метод main с необходимыми атрибутами в нем намеренно отсутствует.

Класс Hello спроектирован только для применения другими классами.

```
class UseHello{

public static void main(String[] args){
 Hello hello=new Hello();
 System.out.println(hello.getHello());
 }
}
```

Здесь в методе `main` создается объект – временная копия – класса `Hello`:

```
Hello hello =new Hello();
```

И вызывается метод `getHello` этого объекта через переменную `hello`, указывающую на него:

```
 hello.getHello();
```

Результат работы программы класса `UseHello` полностью совпадает с результатом `HelloWorld`, но возможности объектно-ориентированной версии гораздо шире.

Дополним действия метода `main` класса `UseHello` и изменим значения полей переменной `hello`:

```
class UseHello{

public static void main(String[] args){
 Hello hello=new Hello();
      hello.greeting="Good day";
      hello.name="my dear";
 System.out.println(hello.sayHello());
 }
}
```

И результат работы программы `UseHello` будет другим:

```
$ java UseHello
Good day, my dear!
```

1.3 Задания

Для изучающих язык Java

1. Используйте аргументы командной строки для установки значения поля hello.name. Добавьте классу Hello метод getHelloTo(String name). Объясните результат.
2. Составьте проект класса, представляющего студенческий билет.
3. Ознакомьтесь с историей создания и развития языка Java. С какой целью создавался этот язык и каким было его первоначальное название?

Для изучающих другой язык программирования

1. Классифицируйте язык программирования, избранный вами для изучения.
2. Определите необходимое для разработки ПО.
3. Напишите и научитесь исполнять программу, подобную `HelloWorld` на Вашем ЯП.
4. Составьте отчет о проделанной работе.

1.4 Итоги

• Определение класса языка программирования позволяет получить сведения, необходимые для начала его изучения.

• Эти сведения дают возможность узнать, как организовать окружение для разработки программ на этом языке.

• Выполнение примера программы подтверждает работоспособность окружения.

• Разбор текста примера дает начальное представление об организации программ для изучаемого ЯП.

2 Лексические элементы языка

2.1 Пояснение

На всех языках программирования программы записываются символами: буквами и цифрами. Какие символы разрешено использовать в ЯП и каким правилам нужно следовать в написании программ, определяет алфавит языка программирования.

Следующий по уровню сложности элемент – комбинации символов, распознаваемые как законченные части инструкций языка – лексемы. Знание типов лексем и правил их записи в тексте программы позволит вам читать программы, напи-

санные другими программистами, и задавать правильные вопросы по их устройству.

2.2 Лексические элементы языка программирования Java

Алфавит языка Java состоит из букв, десятичных цифр и специальных символов. Буквами считаются латинские буквы (кодируются в стандарте ASCII), буквы национальных алфавитов (кодируются в стандарте Unicode-2, кодировка UTF-16).

Лексемы Java с формальной точки зрения можно разделить на 5 различных групп: ключевые слова, разделители, комментарии, идентификаторы и буквальные константы.

Ключевые слова

Ключевые слова: слова (символьные последовательности), имеющие специальное значение в языке. В Java **не допускается использование ключевых слов в качестве идентификаторов**. Некоторые слова являются зарезервированными: они не имеют специального значения, но использовать их качестве идентификаторов нельзя.

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

*не используется; **добавлено в 1.2; ***добавлено в 1.4; добавлено в 5.0
В ключевых словах используются только символы ASCII.

Разделители

Пробельные символы

Пробельные символы: пробел, табуляция, новая строка. Разделяют лексемы.

Скобки

- {} выделяют блоки операторов;
- () отделяют списки параметров методов и используются для управления порядком выполнения операций в выражениях;
- [] используются для выделения индекса элементов массива, а также для описания массивов.

Знаки препинания

Знаки препинания:

- ; --точка с запятой – разделяет операторы;
- . –-точка - используется как разделитель частей путевых имен (но имеет и другое значение);
- , --запятая - разделяет элементы в списках;
- : --двоеточие - отделяет метки от исполняемых операторов.

Знаки встроенных операций

Знаки встроенных операций (и комбинации из них): +, -, /, *, <,>%,^, &, |, ~, !, ?:, = и знак . (точка) можно рассматривать как разделители операндов в арифметических, логических и управляющих выражениях.

2.2.1 Комментарии

Комментарии - фрагменты программы, игнорируемые компилятором. Нужны программистам, а не компьютеру. В Java применяются 3 вида комментариев: однострочный (начинается последовательностью //), многострочный (заключается между последовательностями символов /* и */) и комментарий в стиле javadoc (многострочный комментарий, начинающийся последовательностью /**). Не допускается вложение комментариев друг в друга.

2.2.2 Идентификаторы

Идентификаторы - имена переменных, классов, интерфейсов и пакетов, названия методов. Могут состоять из любых алфавитно-цифровых последовательностей, не начинающихся с цифры. В Java применяется ряд соглашений, регламентирующих использование символов верхнего и нижнего регистра в именованиях. Например, все имена классов принято начинать символом верхнего регистра, а названия публичных методов составлять из строчных букв.

2.2.3 Буквальные константы (литералы)

Буквальные константы - лексемы, передаваемые в программу без изменений, буквально. Правила записи буквальных констант зависят от типа константы. В Java используются буквальные константы числовых типов - целых и вещественных; символьные и строковые буквальные константы.

Тип буквальной константы должен быть однозначно определен по способу ее записи в тексте программы.

- ✓ Целый тип (int): последовательность цифр, не разделяемая точкой, трактуется как буквальная константа типа int, записанная в десятичной системе счисления. Если число начинается нулем, оно трактуется как восьмеричное; если после нуля записывается символ x или X, то число считается целым шестнадцатеричным, и в нем допускаются шестнадцатеричные цифры a,b,c,d,e и f. Примеры: 34, 025, 0xa1fdce.
- ✓ Целый расширенный тип (long): правила те же, что и для целого типа, но в конце числа помещается буква l или L.
- ✓ Вещественный тип (double): допускает только десятичную систему счисления; любое число, содержащее десятичную точку, трактуется как число типа double. Воспринимается и запись числа в экспоненциальной форме, например 0.543e10.
- ✓ Вещественный укороченный (float) - в конце числа с десятичной точкой помещается буква f или F.
- ✓ Символьный (char) - один символ, окруженный апострофами, например, 'a' или эскейп-последовательность unicode-2, также в апострофах. Например, '\u0432'. Как управляющие трактуются эскейп-последовательности:

  1. `'\n'` – перевод строки
  2. `'\f'` – новая страница
  3. `'\t'` – горизонтальная табуляция
  4. `'\r'` – возврат каретки
  5. `'\b'` – возврат на одну позицию
  6. `'\\'` - backslash
  7. `'\''` - апостроф
  8. `'\"'` - кавычка
  9. `'\d','\xd','\ud' – d` - число: восьмеричное, шестнадцатеричное, символ Unicode

- ✓ Логический тип представлен двумя буквальными константами `true` и `false`.
- ✓ Буквальные константы строкового типа описываются производным типом - классом `java.lang.String`. Они распознаются по окружающим их двойным кавычкам, и могут содержать любые последовательности любых символов из алфавита, принятого в Java. Например, `"Hello, world!"` или "!*%ь,Я]\n".
- ✓ Буквальных констант других типов не бывает.

2.3 Задания

Для изучающих Java

Определите все лексические элементы в следующем фрагменте кода:

```
    public static String toString(int i, int radix) {

            if (radix < Character.MIN_RADIX || radix > Charac-
ter.MAX_RADIX)
        radix = 10;

    /* Use the faster version */
    if (radix == 10) {
        return toString(i);
    }

    char buf[] = new char[33];
    boolean negative = (i < 0);
    int charPos = 32;

    if (!negative) {
        i = -i;
    }

    while (i <= -radix) {
        buf[charPos--] = digits[-(i % radix)];
        i = i / radix;
    }
    buf[charPos] = digits[-i];

    if (negative) {
        buf[--charPos] = '-';
    }
```

Не прибегая к компиляции, найдите нарушения правил записи лексических элементов Java в следующих фрагментах кода:

```
1) Class lexeme{}
2) int class=5;
3) if ( a>b) then b:=2;
4) For(int if=2; if<3; if++)
```

Для изучающих другой язык программирования

1. Составьте описание алфавита и лексем изучаемого языка программирования.
2. Сформулируйте правила обращения с ключевыми словами для вашего языка.
3. Придумайте программу-пример нарушения правил записи лексем в изучаемом языке и опишите реакцию на них компилятора и/или исполняющей системы.

3 Встроенные типы данных

3.1 Пояснение

Данные разных типов в двоичном коде записываются компьютером по разным правилам. Определение типа данных, по сути, есть указание правил обращения с предметом – переменной или литералом – этого типа. То есть, определение типа для вычислительной системы обязательно. Однако оно может быть по-разному организовано: в одних языках программирования тип всех объектов должен быть определен заранее, в других – формального определения типа не предусмотрено, но тип объекта определяется по контексту. Первый способ называется статической типизацией, второй – динамическим определением типа.

Что означает понятие «встроенный» для типа? Встроенный – значит заведомо присутствующий в языке, не требующий определения, в противоположность типам, определяемым пользователем. Здесь слово «пользователь» нужно понимать как «пользователь языка программирования», то есть, программист.

Другие названия встроенных типов: примитивные, базовые, простые.

3.2 Типы данных Java

Java относится к языкам со строгой статической типизацией. То есть, типы всех объектов должны быть явно определены при составлении программы до начала использования этих объектов. Это значит, что локальная переменная (переменная, описываемая внутри метода) должна быть описана перед выражением, в котором она будет использоваться.

Типы данных Java отличаются стабильностью: их реальная значимость не зависит от аппаратной платформы, что обеспечивается наличием «посредника» – виртуальной машины, интерпретирующей байт-код в командах конкретного процессора.

Числовые типы

Целочисленные

- `int` - знаковый целый, 32 бита;
- `long` - знаковый расширенный целый, 64 бита;
- `short` - знаковый целый, 16 битов, big-endian;
- `byte` - знаковый целый, 8 битов;

С плавающей точкой

- `double` - вещественный (с плавающей точкой), 64 бита;
- `float` - вещественный укороченный, 32 бита;

Символьный

- `char` - символьный - беззнаковый целый, 16 битов, трактуется как символ Unicode-2;

Логический

`boolean` – логический, представлен двумя буквальными константами `true` и `false`.

3.3 Приведение встроенных типов

Под приведением типов понимается изменение типа объекта, если он участвует в действиях с объектами другого типа. Например, нужно сложить целое число с вещественным, или записать результат целочисленного деления в переменную типа `float`.

3.3.1 Приведение типов в Java

Существуют два способа изменения типов объектов: неявный, выполняемый автоматически, и явный, требующий указаний программиста. Это определяется тем, что тип может изменяться в сторону расширения, например, из `int` в `long`, и в сторону сужения, например из `int` в `byte`. Расширяющие преобразования типов выполняются автоматически, в случае сужающего преобразования необходимо явное указание преобразования типа.

Примеры:
расширяющее автоматическое

```
int c=5;
double f=c++;
```

сужающее явное

```
double d=2.76543;
int h=(int) d/2;
```

Автоматическое приведение типа выполняется по следующим правилам:

1. тип всех операндов выражения расширяется до самого "широкого" типа, участвующего в выражении;
2. типов byte и short не существует, все переменные этих типов в выражениях автоматически приводятся к int;

Важное замечание: тип boolean несовместим с числовыми типами, значения true и false не заменяются никакими числами.

3.4 Задания

Для изучающих Java

1. Определите максимальное и минимальное значение для всех целочисленных типов языка
2. Определите тип выражений:

```
3 + 2
5.1 -6
45L +'a'
```

Для изучающих другой язык программирования

1. Составьте описание правил типизации и типов данных изучаемого языка программирования. Сравните их с типами Java.
2. Опишите правила приведения типов (если таковые имеются) для вашего языка программирования.
3. Составьте краткую памятку по этой теме.

4 Встроенные операции и вычисление выражений

4.1 Пояснение

Встроенные операции – это процедуры, входящие в язык программирования. Они, как правило, удобно и просто оформлены, так, чтобы обращение к ним было максимально кратким и не вызывало никаких затруднений. В результате, обращение к этим процедурам скрывается символом-знаком операции, таким как + (плюс) или – (минус), и начинающий программист не всегда понимает, что на самом деле скрывается за этим «простым» действием.

В отношении встроенных операций языки программирования довольно единообразны. Однако, в одних языках какие-то операции могут отсутствовать, а один и тот же знак в разных языках может обозначать различные операции или иметь дополнительное значение.

Выражение – это последовательность переменных и/или буквальных констант, соединенная (или разделенная, как удобнее) знаками встроенных операций. Выражения, как правило, являются частью какой-либо законченной инструкции (оператора).

Вычисление выражений выполняется в соответствии с приоритетом операций с применением неявного приведения типов в порядке приоритет – приведение. То есть, сначала выделяются подвыражения в соответствии с приоритетом операций, а потом для операндов подвыражения выполняется приведение типов.

4.2 Встроенные операции Java

4.2.1 Арифметические операции

Арифметические операции выполняются с операндами числовых типов и производят результаты числовых типов.

Знак	Название	Результат
+	Бинарное сложение*	Сумма окружающих знак операндов
-	Бинарное вычитание или унарный минус	Разность левого и правого операндов, или противоположное значение
*	Умножение	Произведение окружающих знак операндов
/	Деление	Частное от левого операнда по правому
%	Остаток от деления	Остаток от деления нацело левого операнда на правый
++	Унарная операция инкремента	Увеличивает значение операнда на единицу
--	Унарная операция декремента	Уменьшает значение операнда на единицу

*Знак + имеет дополнительное значение (перегрузку) для символьных строк, где обозначает процедуру соединения (конкатенации) двух строк в одну. Это единственная перегруженная операция в Java. Если один из операндов строкового «сложения» не является строкой, его преобразование в строку происходит автоматически.

4.2.2 Операции сравнения

Операции сравнения производятся над операндами числовых типов, но порождают результаты логического типа.

Знак	Название	Результат
>	Больше	Левый операнд больше правого?
>=	Больше или равно	Левый операнд не меньше (больше или равен) правого?
<	Меньше	Левый операнд меньше правого?
<=	Меньше или равно	Левый операнд не больше (меньше или равен) правого?
==	Равно	Операнды равны?
!=	Неравно	Операнды не равны?
?:	Селектор выражений	Усеченное если-то-иначе для выражений

Подобие операций сравнения для производных типов выполняется **операцией instanceof**: она определяет принадлежность объекта к указанному классу. Производит результат логического типа.

Пример применения селектора выражений

```
class Expressions {

 public static void main(String[] args){
//Генератор случайных чисел:
 java.util.Random rand=new java.util.Random();
 int x=rand.nextInt(10);
 int y=rand.nextInt(10);
//Выражение выбирается в зависимости
// от выполнения условия:
 int action= x > y ? x-y : x+y;
 System.out.println("x="+x+" y="+y + " action="+action);
 }
}
```

4.2.3 Побитовые операции

Побитовые операции применимы только к числовым операндам целых типов, производят результаты также числовых целых типов.

Знак	Название	Результат
~	Унарное побитовое отрицание	Заменяет все разряды в машинном представлении числа на противоположные
|	Бинарное побитовое сложение	Все разряды операндов "складываются" между собой: 1|1 = 1,1|0=1,0|0=0,0|1=1
&	Бинарное побитовое умножение	Все разряды операндов "умножаются" между собой: 1&1 = 1,1&0=0,0∓0=0
ˆ	Бинарное побитовое исключающее сложение	Тоже, что и |, кроме 1 ˆ 1= 0
>>	Сдвиг вправо с сохранением знака	Все разряды левого операнда перемещаются вправо на число позиций, заданное правым операндом. Освобождающиеся слева позиции заполняются знаковым разрядом.
>>>	Сдвиг вправо без сохранения знака	Освобождающиеся позиции заполняются нулями.
<<	Сдвиг влево	Все разряды левого операнда перемещаются влево на

		число позиций, заданное правым операндом. Освобождающиеся справа позиции заполняются нулями.

4.2.4 Логические операции

Логические операции применимы только к операндам логического типа, производят результат логического типа и ... используют те же символы, что и побитовые операции.

Знак	Название	Результат
!	Унарное логическое отрицание	Заменяет значение операнда на противоположное
|	Логическое ИЛИ	Логическая "сумма" двух высказываний
&	Логическое И	Логическое "произведение" двух высказываний
^	Логическое исключающее ИЛИ	То же, что и |, кроме true^true==false
||	Укороченное логическое ИЛИ	То же, что и |, но без гарантии вычисления правого операнда
&&	Укороченное логическое И	То же, что и &, но без гарантии вычисления правого операнда

4.2.5 Операция присваивания

Операция присваивания, (знак **‘=’),** сохраняет результат выполненного действия в переменной. Широко используется сохранение измененного значения переменной на месте старого значения переменной. Например, `int x=2; x=x+2;` означает, что измененное значение переменной `x` будет сохранено в ней же, и это действие может быть записано так: `x+=2`. Возможно сочетание знака ‘=’ со знаком любой из описанных выше операций.

Изменение значения на единицу с последующим сохранением измененного значения выделено в отдельные унарные операции **инкремента (++)** и **декремента (--)**. Пример: `int x=2; x=x+1; int y=x;` может быть записано как `int x=2; int y=++x;` знаки операций инкремента и декремента могут стоять как после операнда, так и до него, в зависимости от того, какое значение операнда (начальное или уже измененное) следует использовать в выражении.

4.2.6 Операция разыменования

Знак **. (точка)** означает операцию разыменования – получения значения по адресу (косвенный доступ). Адресом в случае Java является имя – имя класса или имя объекта. Разыменование применяется для получения значений переменных,

принадлежащих объектам или классам, и для передачи управления в подпрограммы - вызовов методов объекта или класса.

4.2.7 Приоритет встроенных операций

Встроенные операции при вычислениях выражений с числом операций более одной выполняются в порядке слева направо в соответствии с приоритетом. Сначала выполняются все операции высшего приоритета, затем приоритета уровнем ниже, и так далее. Порядок выполнения операций можно изменить с помощью круглых скобок.

↓ **Высший приоритет**

1.	**( ) [ ] .**
2.	**++ -- ~ !**
3.	*** / %**
4.	**+ -**
5.	**>> >>> <<**
6.	**== !=**
7.	**&**
8.	**^**
9.	**|**
10.	**&&**
11.	**||**
12.	**= или op=** **op – знак операции**

↑ **Низший приоритет**

4.2.8 Примеры

1. Вычисление выражения, состоящего из буквальных констант целого типа:

```
class ShowExpressions{
public static void main(String[] args){
   System.out.println("2+3+4=" + (2+3+4));
}
```

```
}
```

Откомпилируем класс и выполним программу (если, конечно, компиляция пройдет успешно.)

```
javac ShowExpression.java && java ShowExpression
2+3+4=9
```

2. Вычисление выражения из буквальных констант разных типов: дополним класс ShowExpression инструкцией

```
System.out.println("2+1.5=” + (2+1.5));
```

К результату выполнения добавится строка:

```
2+1.5=3.5
```

3. Вычисление выражения из буквальных констант с операциями разных приоритетов (добавить инструкции):

```
System.out.println(“3*2 + 0.5=”+(3*2+0.5));
System.out.println(“3/2+0.5=”+(3/2+0.5));
```

Добавят к результату

```
3*2+0.5=6.5
3/2+0.5=1.5
```

Если последний результат удивляет, вспомним порядок применения правил «приоритет-приведение типов» при вычислении выражений.

4.3 Задания

Для изучающих Java

Вычислите выражения, не прибегая к программированию:

1. 2.75-1/2
2. -24>>4
3. 'a'+10

Включите вычисление этих выражений в класс ShowExpression и проверьте свои ответы. Исправьте ошибки и включите в текст класса комментарии, объясняющие результат.

Для изучающих другой язык программирования

1. Составьте краткий отчет-памятку по встроенным операциям, приоритетам операций и правилам вычисления выражений для изучаемого языка.
2. Переведите данные в этом разделе примеры на изучаемый язык программирования.
3. Дополните набор примеров по своему усмотрению.

5 Управляющие конструкции

5.1 Пояснение

Управляющие конструкции позволяют программисту влиять на порядок выполнения операторов. Вообще говоря, операторы внутри одной группы выполняются последовательно (для языков императивного типа). То есть, в том порядке, в котором они были записаны. Управляющие конструкции позволяют выбирать операторы для выполнения в зависимости от условия (ветвление), организовывать циклы (повторения группы действий) и строить переключатели (ветвление по набору условий), передавать управление другой группе операторов и реагировать на ошибки времени исполнения. Условно разделим управляющие конструкции на "локальные управляющие конструкции" и "конструкции, связанные с передачей управления".

5.2 Локальные управляющие конструкции Java

5.2.1 Ветвление (условный оператор)

Разделяет инструкции, выполняемые в зависимости от условия: при выполнении условия выполняется одна группа операторов, в противном случае - другая. Условие задается выражением логического типа. Общий синтаксис конструкции ветвления следующий:

```
if (выражение_логического_типа)
   { набор инструкций для случая true }
else
{ набор инструкций для случая false }
```

Пример ветвления

```
class IfThenElse{
public static void main(String[] args){

int money=(args.length > 0) ? Integer.parseInt(args[0]) : 0;
    if (money>0)
     System.out.println("Деньги есть: "+money+ "рублей.");
    else
     if (money==0)
          System.out.println("Денег нет.");
     else
          System.out.println("Вы должны мне "+
           (-money) + " рублей.");
    }
    }
```

5.2.2 Переключатель

Условные операторы можно вкладывать друг в друга, но при нескольких вложениях конструкция начинает выглядеть громоздко. При выборе из многих возможных значений удобнее использовать другую форму ветвления - переключатель, который записывается так:

```
switch (анализируемое_выражение){
  case значение_1: набор_операторов_1;
  case значение_2: набор_операторов_2;
  ..................................
  case значение_n: набор_операторов_n;
default:набор_операторов_по_умолчанию;
}
```

Тип "анализируемого_выражения" ограничивается базовыми типами, совместимы с типом `int: byte,short,char`, перечислимыми типами `enum`, классами `String` и классами-оболочками `Integer, Byte, Short, Character`. Все "`значения_?`" обязаны быть константами - буквальными или именованными.

5.2.3 Пример переключателя

```
class Switch{
public static void main(String[] args){
switch(args.length){
  case 0:
    System.out.println(
"Применение: java Switch целое_число");
 System.exit(2);
}
int n=Integer.parseInt(args[0]);
System.out.print(n+" - ");
switch(n){
  case 1: System.out.println("пн, работа"); break;
  case 2: System.out.println("вт, работа"); break;
  case 3: System.out.println("ср, работа"); break;
  case 4: System.out.println("чт, работа"); break;
  case 5: System.out.println("пт, работа"); break;
  case 6: System.out.println("сб, отдых"); break;
  case 7: System.out.println("вс, отдых"); break;
  default: System.out.println("Нет такого дня недели.");
  }
}
}
```

Обратите внимание на оператор `break;` в конце каждого набора инструкций. Он необходим для выхода из блока switch после выполнения группы; без оператора `break` все следующие после подходящего `case` группы тоже будут выполняться. То есть, в Java переключатель без искусственного прерывания «проваливается».

5.2.4 Операторы цикла

В Java оператор цикла представлен 4-мя формами.

Оператор for:

```
for(блок_инициализации;условие_продолжения;блок_итераций )
    { группа_операторов; }
```

Группа_операторов(тело цикла) будет повторяться до тех пор, пока условие_продолжения (выражение логического типа) имеет значение true.

Пример цикла for

```
class For{
public static void main(String[] args){
for(int i=0, k=-10; i+k != 0; i++,k++ )
 System.out.println("i="+i + " k="+k);
}
}
```

Оператор for в стиле foreach.

Отличается от for в классической форме другой записью заголовка:

```
for(переменная:объект_типа_Iterable)
{ группа_операторов; }
```

Объектом_типа_Iterable может быть, например, массив.

Пример цикла for в стиле foreach

```
class ForEach{
public static void main(String[] args){
for( String s : args )
 System.out.println( s + " has length "+s.length());
}
}
```

Цикл while.

```
while(выражение_логического_типа)
```

```
{ группа_операторов; }
```

Пока `выражение_логического_типа` имеет значение `true`, группа_операторов будет выполняться.

Пример цикла while

```
import java.util.Random;

class While {
public static void main(String[] args) {
Random rand=new Random();

int num=rand.nextInt(10);
while( num > 2 ){
 System.out.println("num=" + num );
 num--;
}
}
}
```

Цикл do-while.

Иногда требуется гарантировать хотя бы однократное выполнение тела цикла. Это достигается с помощью формы do-while, в которой условие продолжение проверяется после выполнения тела цикла.

```
do { группа_операторов; }
while(выражение_логического_типа);
```

Пример цикла do-while

```
import java.util.Random;

class DoWhile {
public static void main(String[] args) {
//Генератор случайных чисел
java.util.Random rand=
               new java.util.Random();

int num=rand.nextInt(10);
do
{
 System.out.println("num=" + num );
 num--;
}while( num > 2 );
```

```
}
}
```

5.2.5 Метки, операторы break и continue

В Java нет оператора безусловного перехода в произвольное место программы к метке (`goto`). Метки ставятся только в заголовках циклов и переключателей для операторов `break` и `continue`. Оператор `break` прекращает выполнение группы операторов в переключателе или цикле, отмеченном меткой, и передает управление оператору, следующему за отмеченным блоком. Оператор `continue` прекращает выполнение группы операторов в теле отмеченного цикла, вызывая переход к следующему шагу цикла. Операторы `break` и `continue`, не имеющие меток, действуют в группе операторов, ограниченной ближайшими фигурными скобками.

Пример перехода по метке

```
class GetPrimes{
public static void main(String[] args){
//Верхняя граница  интервала поиска по //умолчанию
int n=20;
//Границу можно задать в командной строке
if(args.length==1) n=Integer.parseInt(args[0]);


W: for(int i=1; i<n; i+=2){

 for(int j=2; j<=i/2 ; j++){
      if(i%j==0) continue W;
 }
 System.out.print(" "+i);
   }

System.out.println("");
}
}
```

5.3 Конструкции передачи управления Java

5.3.1 Методы: описание, формальные параметры

В Java нет независимых процедур и функций; подпрограммами в этом языке программирования являются методы различных классов.

Метод - именованная группа операторов, связанная с именем класса или экземпляра (объекта) класса. Описание метода является частью описания класса и размещается в теле класса.

Описание метода состоит из заголовка и тела; тело метода состоит из группы операторов, хотя бы с одним оператором возврата управления. Заголовок метода имеет вид:

```
[модификаторы] тип_возвращаемого_значения
имя_метода([список_формальных_параметров]) [декларация throws]
```

Список_формальных_параметров является перечнем описаний локальных (ограниченных телом метода) переменных.

Тип возвращаемого значения описывает тип результата работы метода – тип аргумента оператора return. Отсутствие возвращаемого значения описывается ключевым словом void.

Пример описания методов в классе Hello

```
class Hello{
String greeting="Hello";
String name="World";

void sayHello(){
        System.out.println(greeting+", " + name +"!");
}

String getHelloTo(String name){
        return greeting+", " + name +"!");
}

String getHello(){
        return greeting + ", " + name +"!";
}
}
```

5.3.2 Вызов метода: передача фактических параметров

Применение метода происходит тогда, когда другая программа обращается к нему с уже определенными значениями аргументов:

```
class UseHello{
public static void main(String[] args){
 Hello hello=new Hello();
 String current=hello.getHello();
 System.out.println(current);
 String my_hello=hello.getHelloTo("Ольга Львовна");
 System.out.println(my_hello);
}
}
```

При обращении к методу на место формальных аргументов подставляются конкретные значения соответствующих типов:

```
String my_hello=hello.getHelloTo("Ольга Львовна");
```

Вызов метода класса или объекта является операцией явной передачи управления.

5.3.3 Возвращение из метода, оператор return

По завершении метода управление возвращается к той инструкции, откуда был произведен вызов. Возврат управления осуществляется оператором `return [возвращаемое_значение]`. Если `возвращаемого_значения` нет, оператор `return` в теле метода может быть опущен. При наличии `возвращаемого_значения` оператор `return` с аргументом обязателен. Тип аргумента оператора `return` должен совпадать с `типом_возвращаемого_значения` в описании метода. В методе может быть использовано несколько операторов `return`.

5.3.4 Аварийное завершение метода, оператор throw

Кроме «штатного» возвращения управления оператором `return`, в Java есть оператор аварийного завершения метода `throw`. «Аварией» считаются условия времени исполнения, при которых метод не может быть выполнен. Наличие оператора аварийного завершения позволяет организовать обработку ошибок времени исполнения в принудительном порядке.

Оператор `throw` сопровождается обязательным аргументом – объектом типа `Throwable`, содержащем информацию об обстоятельствах «аварии».

Оператор `throw` прекращает работу метода и передает управление (и свой аргумент типа `Throwable`) в блок обработки ошибок `catch`, следующий за блоком попытки (`try`) исполнения неудавшегося метода.

Аварийные ситуации делятся на ошибки (`Error`), не требующие действий со стороны программиста, и исключения (`Exception`), обработка которых на программиста возлагается.

5.3.5 Блоки try-catch-finally

Наличие в теле метода оператора `throw` отслеживается компилятором. Компилятор потребует предусмотреть обработку ошибки времени исполнения на месте или добавить в заголовок метода декларацию о возможности нештатного завершения метода. Вызов метода с декларацией `throws` в заголовке должен быть помещен в специальный блок операторов `try`, за которым в обязательном порядке должен следовать блок инструкций на случай нештатного завершения вызова - блок `catch` – и/или блок обязательных операторов `finally`. В блок `catch` передается управление оператором `throw`, если он случился, вместе с его аргументом. За од-

ним блоком try могут следовать несколько блоков catch для обработки исключений различных типов.

```
try{
 потенциально_опасный_код;
}catch(Исключение1 имя_переменной){
 операторы_для_обработки_1;
}catch(Исключение2 имя_переменной){
 операторы_для_обработки_2;
}catch(.............................
.................................
}finally{
 обязательные_операторы;
}
```

Пример блоков try-catch-finally

Рассмотрим программу из трех инструкций, в которой может случиться три ошибки времени исполнения:

```
class ThreeExceptions{

public static void main(String[] args){

  int p=Integer.parseInt(args[0]);
  int q=Integer.parseInt(args[1]);
  System.out.println(p+":"+q+"="+p/q);
}
}
```

К краху этой программы может привести отсутствие или нехватка аргументов командной строки, несоответствие аргумента целому числу и деление на ноль. В следующей версии этой же программы все возможные исключения времени исполнения предусмотрены:

```
class TryCatchFinally{

public static void main(String[] args){

try{
  int p=Integer.parseInt(args[0]);
  int q=Integer.parseInt(args[1]);
  System.out.println(p+":"+q+"="+p/q);
}
catch(ArrayIndexOutOfBoundsException e){
        System.out.println(
```

```
            "Недостаточно аргументов");
    }
    catch(NumberFormatException e){
            System.out.println(
            "Невозможно выполнить преобразование в число "+
            e.getMessage());
    }
    catch(ArithmeticException e){
            System.out.println(
            "Деление на ноль не определено");
    }
    finally{
            System.out.println(
            "Программа успешно завершена");
    }
    }
    }
```

5.4 Задания

Для изучающих Java

1. Напишите программу, правильно формирующую предложение "n дней и m лет", где n и m - случайные целые от 1 до 7.
2. Напишите подпрограмму, определяющую является ли заданное число простым. Напишите программу, демонстрирующую работу этой подпрограммы.
3. Дополните программу из предыдущего задания блоками обработки ошибок времени исполнения: выход за границу массива, недопустимый формат числа.
4. Рассмотрите возможность генерации исключения при проверке на простоту неположительного числа.

Для изучающих другой язык программирования

1. Соберите сведения об управляющих конструкциях вашего языка программирования. Составьте краткую справку обо всех управляющих конструкциях.
2. Изучите вопрос с организацией ошибок времени исполнения в вашем языке.
3. Перепишите программу из класса TryCatchFinally на вашем языке программирования.

6 Встроенные структуры данных

6.1 Пояснение

Под структурой данных будем понимать организованное определенным образом множество однотипных элементов, которое позволяет группировать под одним именем несколько значений, что открывает колоссальные возможности для программистов. Встроенные структуры данных, как и базовые типы, не нужно описывать, они присутствуют в языке изначально. Некоторые ЯП предлагают только массивы; другие же, например, Python, поддерживают большое разнообразие таких структур (списки, кортежи, словари).

6.2 Встроенные структуры данных Java

Java относится к языкам «скупым на встроенные структуры данных», так как располагает только массивами и перечислениями, списки же и словари реализованы там библиотечными классами.

6.2.1 Массивы

Массив - это индексированный набор однотипных элементов ограниченной длины. То есть, переменная-массив объединяет в себе несколько однотипных элементов, обращения к которым осуществляются по индексу (номеру) - неотрицательному целому числу. Число элементов в массиве определяется при его создании и не может быть изменено за время жизни массива. В Java нет принципиальной разницы между массивами данных встроенных и производных типов, правила работы с ними, за небольшим исключением, одинаковы.

6.2.2 Правила обращения с массивами

1. В Java нет ключевого слова, соответствующего понятию "массив", массив обозначается парой квадратных скобок `[]`, следующей после названия типа или имени переменной, вот так:

```
double[] parameters;
int [] numbers;
String args[];
Object[] others;
```

2. Массив создается оператором `new`, так же, как объекты классов. При этом определяется число элементов в массиве, которое потом не поддается изменению:

```
numbers=new int[10];
```

3. Число элементов в массиве - его длину - всегда можно узнать по его полю `length` (сюрприз!), вот так:

```
numbers.length
args.length
```

4. Чтобы обратится к элементу массива, нужно указать индекс этого элемента в квадратных скобках при имени переменной-массива:

```
for( int i=0; i<numbers.length; i++) numbers[i]=i;
```

Нумерация элементов массива начинается с 0. Обратите внимание на различие смысла чисел в квадратных скобках при создании массива и при обращении к элементу массива.

5. Допускаются многомерные массивы. (Размерность массива определяется способом нумерации элемента в нем. Можно считать "номер в строке", а можно "номер в строке, номер в столбце, и номер в колонке"). Размерность массива описывается количеством пар квадратных скобок: `[][]` - двумерный, а `[][][]` - трехмерный.

```
double [][] matrixA = new double[10][10];
matrixA[0][0]=1.5743276;
```

6. При создании многомерного массива достаточно указать длину массива только в первом измерении; длину в следующих измерениях можно определять потом по мере необходимости. Это свойство позволяет создавать многомерные массивы не прямоугольной формы

```
int[][] matrixT=new int[3][];
matrixT[0]=new int[10];
matrixT[1]=new int[5];
matrixT[2]=new int[7];
```

7. Для задания значений элементам массива можно использовать конструктор массива

```
int[] vector=new int[]{1,3,2,8,4,9,0};
```

Длину массива при наличии конструктора лучше не указывать, она будет определена автоматически.

6.2.3 Пример массива

```
class VerySimpleArrayExample {
```

```
 public static void main(String[] args){
 //Генератор случайных чисел:
 java.util.Random rand=new java.util.Random();
 //Описание переменной-массива:
 int [] arr;
 //Создание (выделение памяти) под 10 элементов:
 arr = new int[10];
 //Заполнение массива случайными целыми:
 for( int i=0; i<arr.length; i++) arr[i]= rand.nextInt(10);
 //Печать элементов в строчку (только в цикле!)
 for (int k : arr )
        System.out.print(k + " ");
 //Перевод строки:
 System.out.println();
 }
}
```

Более сложные примеры, детально раскрывающие приемы применения массивов можно найти на сайте курса.

6.2.4 Перечисления

Перечисления представляют собой наборы именованных констант. Надо заметить, что введены они были только в 5-ой версии языка, примерно, в 2003 году, то есть лет 10 Java обходилась без них. Так как перечисления встраивались в уже сложившийся язык, выглядят они несколько искусственно, но польза от них, безусловно, есть.

Правила работы с перечислениями

1. Перечисления создаются как самостоятельные программные единицы описанием enum:

```
enum Apple{
 Idared, Golden, Fuji, Antonovka,
     GrennySmit, Jonagold
 }
```

и компилируются, как классы.

2. Затем используются в других классах виде переменных типа описанного перечисления:

```
Apple ap=Apple.Fuji;
```

Диапазон значений переменной ap предопределен. С первого взгляда выглядит не слишком умно, но ничего лучше для описания дней недели или месяцев года пока не придумали..

3. Приятный сюрприз в перечислениях тоже есть - это методы `values()` и `valuesOf()`. Метод `values()` выдает набор значений перечисления в виде массива

```
Apple apples[]=Apple.values();
```

а метод `valueOf(String value)` возвращает значение константы, соответствующее строке value:

```
ap=Apple.valueOf("Idared");
```

Кроме того, перечисления отлично выглядят в печатном виде и идеально подходят для использования в переключателях.

Пример: перечисление как ключ коллекции-карты

```
enum DaysOfWeek{
 Понедельник, Вторник, Среда, Четверг,     Пятница, Суббота,
Воскресенье
}

enum Seasons{
        Зима,Весна,Лето,Осень
}

class UseEnum{
public static void main(String[] args){
  Seasons[]  ss=Seasons.values();
  for(Seasons s : ss )
         System.out.print(s+" ");
  System.out.println();
  DaysOfWeek[] days
           =DaysOfWeek.values();
  for(DaysOfWeek d : days )
         System.out.print(d+" ");
  System.out.println();
//класс Date представляет дату:
  java.util.Date now=
           new java.util.Date();
//Нумерация дней недели требует коррекции
  int numOfDay=now.getDay() > 0 ?
              now.getDay()-1 : 6;
//Зима включает месяцы с номерами 11,0 и 1:
int numOfSeason=
      (now.getMonth()!=11) ?
             now.getMonth()/3 :0;
```

```
        System.out.println(
                 "Сегодня день недели - "
                 +days[numOfDay]+
                 ", а время года - "
                    +ss[numOfSeason]);
        }
    }
```

6.3 Задания

6.3.1 Для изучающих Java

1. Напишите подпрограмму, вычисляющую произведение двух совместимых матриц и программу, вызывающую ее. Предусмотрите возможные ошибки времени исполнения.
2. Опишите перечисление, соответствующее дням недели и программу - расписание, выдающую расписание занятий на случайно выбранный день недели.

6.3.2 Для изучающих другой язык программирования

1. Составьте описание встроенных структур данных в вашем ЯП.
2. Напишите подпрограмму, вычисляющую произведение двух совместимых матриц и программу, вызывающую ее. Если возможно, предусмотрите обработку возможных ошибок времени исполнения.

7 Пользовательские типы данных

7.1 Пояснение

Возможность определять собственные типы данных и оперировать ими так же, как встроенными типами, массивами и перечислениями – очень важное свойство языка программирования. Этой возможности может не быть совсем, она может предоставляться как небольшое дополнение к встроенным типам, а может составлять главную черту языка программирования. Java относится к последним: основу этого языка программирования составляет создание пользовательских типов данных в виде классов и интерфейсов.

Обучение программированию на Java в значительной степени состоит в постижении правил описания пользовательских типов данных и правил их последующего использования.

7.2 Пользовательские типы Java

В Java определяются два вида пользовательских типов – классы и интерфейсы. Классы представляют типы данных с реализацией методов, интерфейсы явля-

ются «пустыми» типами, перекладывающими реализацию методов на классы, присоединяющие (имплементирующие) их.

7.2.1 Принципы объектно-ориентированного программирования

Объектно-ориентированное программирование основывается на применении трех методологий: инкапсуляция, наследование и полиморфизм. Инкапсуляцией называется соединение (встраивание) в объекте (экземпляре класса) данных и методов. Наследование – способ включить во вновь создаваемый класс код существующего класса, чтобы сочетать использование готового кода с его дополнением и модификацией. Полиморфизм – способность объекта менять тип в рамках совместимости, проявляя многообразие свойств. Правила описания классов и интерфейсов проще понять, имея в виду принципы ООП.

7.2.2 Правила описания классов

Описание класса состоит из заголовка и тела класса. В заголовке должно присутствовать ключевое слово class и название класса - произвольный идентификатор, уникальный в своей области видимости. В заголовке могут присутствовать модификаторы (public, abstract и final) и декларации (extends и implements). Тело класса заключается в фигурные скобки, внутри помещаются описания переменных и методов. Методы - именованные группы операторов - имеют свои заголовки и тела, внутри которых также могут описываться переменные. Переменные, описываемые внутри тела метода, являются локальными переменными метода и недоступны для других элементов класса.

Кроме переменных и методов, в описании класса могут находиться описания конструкторов и неименованные блоки операторов - блоки инициализации. Конструктор - блок операторов с предопределенным именем (именем класса). Операторы конструктора выполняются при создании экземпляров классов, в частности, ими может выполняться инициализация данных объекта. Конструкторы являются необходимой частью описания класса. Методов же в описании класса может и не быть.

По умолчанию переменные и методы класса являются *динамическими*, то есть, принадлежащими объектам. Иными словами, их невозможно использовать, не создав предварительно экземпляр класса. Элементы класса, принадлежащие непосредственно классу, помечаются модификатором static. Модификатор static могут иметь переменные, методы, блоки инициализации и вложенные классы (описания классов можно размещать внутри других классов и даже элементов класса). Статические элементы существуют в единственном экземпляре; изменение статической переменной одним объектом приведет к ее изменению в другом объекте. Для обращения к статическим элементам класса достаточно указать имя класса.

Пример описания класса

По аналогии со строкой спроектируем и опишем класс Word (Слово), стараясь совмещать формальные правила описания класса и практический смысл создания нового типа данных.

```
//заголовок класса и открывающая скобка
class Word{
//Поля: данные (информация), хранимые объектами-словами

//буквы слова
char[] letters;

//Методы: процедуры, которые можно проделывать со словами

//Вычислить длину слова
int length(){
      return letters.length;
}
//Напечатать слово

void print(){
      System.out.println(new String(letters));
}

//Представить в виде строки
String asString(){
      return new String(letters);
}

//Конструкторы: процедуры, выполняемые при создании объектов

//Конструктор слова из массива типа char
Word(char[] letters) throws Exception{
/*
* Слово - строка, содержащая только буквы.
* Если хотя бы один символ в массиве letters окажется не буквой,
* создать объект не удастся.
*/
   for(char c : letters) {
    if(! Character.isLetter(c)) throw new Exception("Is not a
letter: "+c);
   }
      this.letters=letters;
 }
//пока все
     }
```

Вот и все, новый тип данных готов. Помним, класс - это не программа. Использовать его можно в методах других классов в виде объектов, например так:

```
Word myName=new Word(new char[]{'О','л','я'});
int l=myName.length();
```

Следующий уровень. Возможны многочисленные детали и уточнения: модификаторы (доступа, принадлежности, изменяемости), совмещение методов, описание блоков инициализации и вложенных классов.

7.2.3 Наследование

Созданный программистом тип данных можно использовать как материал для дальнейшего развития, то есть создать следующий тип на основе уже созданного. Это называется наследование классу (или расширение класса). Наследование позволяет автоматически включить все доступные элементы готового класса в новый тип:

```
class RussianWord extends Word{}
```

Все, больше можно ничего не писать: методы `length()` и `print()`, а также поле `letters` в классе `RussianWord` уже есть. Конечно, обычно так не делают. Если понадобилось расширять класс, значит в нем чего-то не хватало или было сделано не так, как хотелось, то есть, что-то туда все-таки дописывают или что-то замещают. Но не всегда. Пример - классы типа `Exception`. Они, как правило, состоят из одного названия, все остальное наследуется, и очень эффективно.

Наследование создает иерархии совместимых производных типов, то есть, можно связывать объект с переменной совместимого типа, соблюдая правила "снизу вверх", например,

```
Word w=new RussianWord();
```

а потом "восстанавливать" более "низкий" тип:

```
RussianWord rw=(RussianWord) w;
```

Важно: наследовать можно только одному классу. Если в заголовке класса нет декларации `extends`, то он наследует классу `java.lang.Object`. Поэтому все производные типы совместимы (могут быть приведены к нему) с типом `Object`.

Следующий уровень. В наследовании есть свои тонкости: его можно запретить прямо или косвенно, можно экранировать наследуемые компоненты.

7.2.4 Абстрактные классы

Иногда класс разрабатывается исключительно как "основа" для наследования. То есть, требуется не допустить создания объектов этого класса оператором new. Тогда класс объявляют абстрактным, добавляя в его заголовок модификатор abstract. Например, так:

```
abstract class Word{
char[] letters;
int length(){
```

```
     return letters.length;
    }
    void print(){
     System.out.println(new String(letters));
    }
    String asString(){
     return new String(letters);
    }
    Word(char[] letters) throws Exception{
       for(char c : letters) {
        if(! Character.isLetter(c)) throw new Exception("Is not a
letter: "+c);
       }
     this.letters=letters;
     }
    }
```

Как база для наследования, абстрактный класс может вынудить «наследников» определять часть методов. Для этого используются абстрактные методы. Описание абстрактного метода модификатором `abstract` состоит из заголовка и списка формальных параметров, но не содержит тела метода. Наличие в классе одного абстрактного метода требует объявления класса абстрактным.

Пример абстрактного класса как основы для наследования

Представим, что мы задумали создать классы, представляющие кошку (Cat) и собаку (Dog). Класс включает информацию о числе лап и наличии хвоста и метод, воспроизводящий звуки, издаваемые объектом. В перспективе намечается необходимость создания аналогичных классов для коровы, лошади и свиньи (как минимум).

Конечно, нужно использовать наследование, но как? Ни одна версия из «Cat extends Dog» и «Dog extends Cat» не кажется логичной.

В таком случае создается абстрактный класс, представляющий домашнее животное вообще (Pet), а все конкретные представители расширяют его.

```
    abstract class Pet {
    int pows=4;
    boolean hasTail=true;

    abstract void voice();
    }

    class Cat extends Pet{
    void voice() {
      System.out.println("Мяу-мяу!");
```

```
}
}

class Dog extends Pet {
void voice() {
  System.out.println("Гав-гав!");
}
}
```

Продолжать это ряд можно сколько угодно, не вступая в конфликт ни с родительским классом, ни с классами-соседями по иерархии.

7.3 Интерфейсы

Изящная идея с абстрактными классами получила развитие в направлении исправления неудобств, произошедших от отказа от множественного наследования. Интерфейс - это класс, в котором все методы абстрактные (а все поля - статические константы, но это **следующий уровень**). Эта конструкция не приносит никакой пользы в смысле снижения трудозатрат, но определяет тип, не создавая проблем с конфликтами реализаций. Интерфейс описывается почти как класс

```
interface Runnable{
   void run();

}
```

Все методы в интерфейсах по определению public и abstract, указывать эти модификаторы не нужно. Интерфейсы не наследуются, а *имплементируются* классами:

```
class MyClass extends java.awt.Frame implements Runnable{
   ..................................
   }
```

Имплементация – внедрение, выполнение, реализация – интерфейса классом означает наличие в классе методов, описанных интерфейсом. Это дает возможность определять тип экземпляров класса типом интерфейса.

```
MyClass my_class=new MyClass();
Runnable mc=(Runnable) my_class;
```

Имплементировать можно несколько интерфейсов.

7.4 Описание интерфейса

Этот раздел, включающий пункты

1) Правила описания интерфейсов
2) Имплементация интерфейсов
3) Наследование интерфейсов

нужно отнести на **следующий уровень:**

7.5 Задания

Для изучающих Java

1. Найдите в документации описание класса `java.lang.String` и ознакомьтесь с его структурой.
2. Спроектируйте и напишите класс `Complex`, представляющий комплексное число.

Для изучающих другой язык программирования

1. Изучите вопрос об определении пользовательских типов данных в вашем ЯП.
2. Составьте краткое описание значения пользовательских типов для изучаемого языка и правил их создания.
3. Если возможно, опишите пользовательский тип, представляющий комплексное число.
4. Выясните, возможно ли обозначать арифметические действия над экземплярами этого типа знаками встроенных операций.

8 Организация и использование библиотек

8.1 Пояснение

Умение программировать складывается из умения алгоритмизировать, знания языка программирования и знания его библиотек. Способность к алгоритмизации нарабатывается довольно быстро, язык можно выучить по учебникам и справочникам, изучение же библиотек затруднительно – по разным причинам. Очевидно, что программирование эффективно тогда, когда оно позволяет легко использовать ранее написанные программы.

В процедурных языках принято хранение готовых к использованию процедур в виде объектных модулей в статических или динамических библиотеках. Модули из статических библиотек непосредственно включаются в программу при компиляции, модули из динамических библиотек разыскиваются и подключаются к программе при выполнении. Оба способа имеют свои достоинства и свои недостатки: статическое присоединение обеспечивает независимость программы от среды исполнения, но увеличивает ее размер, динамическое позволяет использование одного экземпляра библиотеки множеством разных программ, но ставит программу в зависимость от среды исполнения.

В этом разделе рассматривается техническая сторона устройства библиотек как комплексов уже готовых процедур для новых программ.

8.2 Пакеты – библиотеки классов Java

Организация библиотек в Java по форме от процедурных библиотек сильно отличается, поскольку программной единицей в Java является класс. Библиотеки Java представляют собой группы классов, которые называются *пакетами*.

Среда исполнения Java полностью контролируется разработчиками языка, и все библиотеки в ней используются только динамически. Но так как компилятор работает также в среде исполнения, обязанность разрешения внешних ссылок везде, где возможно, возлагается на него. Результат компиляции - лаконичный малообъемный байт-код классов при многочисленных библиотечных архивах.

Объединение группы классов в пакет решает одновременно несколько проблем: исключает конфликты имен классов, упорядочивает размещение и поиск классов и создает дополнительный уровень защиты классов и их компонентов.

8.3 Полные имена классов

Уникальность имен классов Java суперважна: если в области действия компилятора или интерпретатора оказываются два разных класса с одинаковым именем, использоваться будет тот, который был найден первым. Когда компилятор и интерпретатор по одному имени находят разные классы, исполнение вряд ли окажется успешным. Когда Java-программисты всего мира выбирают имена для нового класса, трудно объяснить каждому из них, какие имена уже заняты.

Принадлежность пакету прибавляет к имени класса имя пакета, что повышает шансы в создании уникальных имен. Если же при организации пакетов их разработчики следуют соглашениям по именованию пакетов по доменному имени, закрепленному за организацией (например, org.apache.ds), то вероятность конфликта имен становится еще ниже.

8.4 Директива package

Принадлежность класса к определенному пакету устанавливается директивой package. Директива package принадлежит файлу, содержащему описание класса, а не классу (один файл может содержать описание нескольких классов). Директива package должна быть первым не закомментированным оператором в файле. Пример библиотечного класса:

```
package java.lang;

public final class String {
.....
```

Или пример того, как поместить свой класс Word в собственный пакет `my.java.works`:

```
package my.java.works;

public class Word{
........
}
```

8.5 Связь имен классов с архивно-файловой структурой хранения

Объявление класса частью указанного пакета требует его размещения в определенной файловой структуре, продиктованной именем пакета. Так файл класса String.class должен находиться в каталоге lang, а каталог lang - в каталоге java. В каком каталоге находится каталог java, уже не имеет значения. Аналогично, файл Word.class должен помещаться в каталоге works, works - в каталоге java, каталог java - в каталоге my. Если вашим текущим каталогом является тот каталог, в котором находится каталог my, вы сможете найти класс Word по его полному имени:

```
$ java my.java.works.Word
```

8.6 Переменная окружения CLASSPATH

Если текущий каталог не содержит каталога my, то разыскать класс my.java.works.Word невозможно. Для решения проблемы поиска путей к классам используется переменная окружения CLASSPATH (язык интерпретатора команд сам является языком программирования, в нем можно определять переменные и задавать им значения). В эту переменную в стиле, принятом в операционной системе, заносится список каталогов, в которых можно поискать классы по их полным именам. К примеру, если каталог my со всеми дальнейшими подкаталогами и файлом класса Word помещается в вашем каталоге /home/your_name/Java, тогда этот каталог и должен быть включен в переменную CLASSPATH:

```
$ export CLASSPATH=/home/your_name/Java:$CLASSPATH
```

Так делается в системе Unix. При работе в Windows переменная назначается иначе:

```
set classpath="C:\\Documents and Set-
tings\your_name\Java\";%classpath%
```

Кроме каталогов, в переменную CLASSPATH можно включать файловые архивы в jar- или zip-формате.

```
$ export CLASSPATH=/opt/java-libs/log4j-1.2.25.jar:$CLASSPATH
```

Имя архивного файла указывается полностью.

При неопределенной переменной `CLASSPATH` компилятор и интерпретатор используют внутреннее значение, включающее jar-файл с классами стандартных

пакетов и все jar-файлы каталога расширений виртуальной машины (`$JAVA_HOME/jre/lib/rt.jar` и `$JAVA_HOME/jre/lib/ext/*.jar`). Переменная `CLASSPATH`, назначенная в командной строке только дополняет внутреннее значение.

8.7 Поиск классов

Поиск классов осуществляется в следующем порядке: **внутри класса → внутри пакета → в каталогах переменной CLASSPATH**

Каталоги и архивы для поиска нестандартных классов можно указать опцией `-classpath` компилятора и/или интерпретатора:

```
$ java -classpath
/home/your_name/Java/ my.java.works.Word
```

8.8 Подключение библиотек

Подключение библиотек в Java сводится к внесению каталога, в котором находится пакет с нужным классом, в пути поиска классов.

8.9 Директива import

Использовать длинные полные имена классов не слишком удобно. Чтобы обеспечить возможность обращаться к классам по их кратким именам, используется директива `import`, дополняющая краткое имя класса до полного:

```
import имя.пакета.имя_класса;
```

или с шаблоном *, означающим «все классы без подкаталогов» (подпакетов):

```
import имя.пакета.*;
```

Пример:

```
import java.util.Random;

class UseRandom{
.......
 Random rand=new Random();
}
```

Директива `import` относится ко всем классам файла, в котором она помещается. Перед директивой `import` может помещаться только директива `package` (комментарии не в счет).

8.10 Пакеты и уровни доступа

Классы объединяются в пакет по смыслу. Например, "самые необходимые" - пакет `java.lang`, "очень полезные" - пакет `java.util`, "все для ввода-вывода" - пакет `java.io` и т.п.

С точки зрения класса, все классы можно поделить на категории - "свои" (из моего пакета), и "чужие" - из других пакетов. В соответствии с таким разделением существуют четыре уровня доступа к классам и их компонентам:

1. "только мое " - `private`
2. "для своих" - устанавливается по умолчанию
3. "для своих и для чужих наследников" - `protected`
4. "для всех" - `public`

8.11 Пакет по умолчанию

Директива `package` не является обязательной. Описание класса можно свободно разместить в файле без директивы `package`. Такие классы автоматически относятся к пакету по умолчанию, который только на первый взгляд ограничен рамками своего каталога.

Классы пакета по умолчанию не имеют защиты пакетного уровня; они создаются, как правило, на начальной стадии обучения или с целью отладки.

8.12 Задания

Для изучающих Java

1. Поместите все ранее написанные исполняемые классы в пакет с именем, содержащим ваш login.
2. Откомпилируйте и запустите на исполнение этот класс с помощью опции -classpath,
3. и корректируя переменную окружения CLASSPATH.
4. Определите на вашем компьютере каталог установки jdk (JAVA_HOME), найдите в нем файл rt.jar и познакомьтесь со структурой этого архива. (команда jar tf rt.jar|less).

Для изучающих другой язык программирования

1. Опишите принципы формирования, распространения и использования библиотек в изучаемом языке программирования.

9 Библиотека поддержки

9.1 Пояснение

Библиотекой поддержки (стандартной, основной библиотекой) языка программирования называют библиотеку, автоматически подключаемую компилятором и исполняющей системой. Процедуры и функции библиотеки поддержки доступны так же, как встроенные операции. Как правило, в библиотеку поддержки включаются подпрограммы первой необходимости; однако понятие "первая необ-

ходимость" в разных языках разное, так как определяется назначением языка. Знание библиотеки поддержки необходимо программисту так же, как знание встроенных элементов языка.

9.2 Пакет java.lang – библиотека поддержки Java

В языке программирования Java роль библиотек выполняют пакеты. Пакет, доступный по умолчанию, называется java.lang. Все классы этого пакета доступны по умолчанию и определяются своими короткими именами. Директива import java.lang.* автоматически добавляется к описанию любого класса (если этого не сделано явно). Пакет java.lang относительно невелик и включает классы и интерфейсы, связанные с конструкциями языка и механизмом функционирования виртуальной машины, а также самые «востребованные» классы общего назначения. Полное описание пакета java.lang, несмотря на его небольшой объем, выходит далеко за рамки данного курса. Мы ограничимся классами и интерфейсами, связанными с поддержкой языковых конструкций и общими базовыми классами.

9.3 Классы пакета java.lang

9.3.1 Класс java.lang.Object

Этот класс является базовым для всех других классов, наследующих ему непосредственно или через цепочку наследования. Отсутствие в описании класса декларации `extends` равносильно объявлению extends `java.lang.Object`. Все методы класса `Object` автоматически включаются в новые классы, и это необходимо учитывать при создании класса-типа. Свойства, унаследованные от `Object`, активно используются интерпретатором Java (виртуальной машиной).

Класс `Object` формирует «скелет» всех классов. Его методы необходимы виртуальной машине для управления объектами в течении их жизненного цикла от создания до освобождения памяти, занятой объектом. Конструктор `Object()` выполняется при создании нового экземпляра любого класса. Незаменяемый метод `getClass()` возвращает объект типа `java.lang.Class`, представляющий класс, экземпляром которого объект является. Метод `toString()` возвращает строку (экземпляр класса `String`) текстового представления объекта. Метод `hashCode()` возвращает идентификатор объекта в виртуальной машине – хэшкод. Метод `equals(Object other)` проверяет равенство текущего объекта объекту `other`. Метод `finalize()` вызывается программой-сборщиком «мусора» – объектов, на которые нет ни одной ссылки. Неизменяемые методы `wait(..)`, `notify()` и `notifyAll()` являются инструментами управления потоками исполнения (thread), необходимыми в многопоточном программировании.

Пример: изменяемые методы класса Object

```
class ObjectDemo{
private static void p(String s){
 System.out.println(s);
}
public static void main(String[] args){
Object o=new Object();
p("Hashcode – уникальный идентификатор объекта:
"+o.hashCode());
Object q=new Object();
p("У другого экземпляра хэшкод другой: "+q.hashCode());
p("Каждый объект способен представить себя в виде строки:
"+o.toString()+" "+q.toString());
p("Объекты можно проверять на равенство:
o.equals(q)="+o.equals(q));
p("Разные объекты класса Object не могут быть равны; равен-
ством может быть только идентичность ссылок:");
Object r=o;
p("Object r=o; o.equals(r)="+o.equals(r));
p("Класс Object определяет метод finalize(), вызываемый сбор-
щиком мусора:");
Object oe=new Object(){
 public void finalize(){
 System.out.println("Qu-qu!");
 }
};
p("Освободим переменную: oe=null;");
oe=null;
p("и вызовем сборщика мусора: System.gc();");
System.gc();

}
}
```

Объекты любых классов могут управлять потоками исполнения, так как наследуют неизменяемые методы класса Object `wait()`, `wait(long millis)`,`wait(Long millis, int nanos)`,`notify()` и `notifyAll()`. Методы `wait(…)` переводят поток исполнения в состояние ожидания, методы `notify()` и `notifyAll()` переводят один из ожидающих потоков в рабочее состояние.

9.3.2 Класс Throwable и его расширения Error и Exception

Класс `Throwable` описывает тип, соответствующий сообщениям об ошибках времени исполнения. Объекты этого (или совместимого с ним) типа являются обязательными аргументами оператора `throw` и только они могут использоваться в качестве его аргументов. Экземпляры этих классов содержат информацию о методе, в котором произошла ошибка времени исполнения и трассировку вызовов – стек передач управления по цепочке вызовов. Его расширения, классы `Exception` и `Error`, служат для разделения причин аварийных завершений на обязательные для обработки и те, которые обработке не подлежат. Так, например, отсутствие файла для чтения данных, описывается исключением (exception), которое программист в силах предусмотреть, а нехватку оперативной памяти для виртуальной машины побороть программными средствами удастся вряд ли и такую «аварию» считают ошибкой (error). Соответственно, оператор `throw` с аргументом типа `Error` не приводит к требованиям дополнительных действий со стороны программиста, а `throw` с аргументом типа Exception требует специальных действий по обработке исключения. Пакет `java.lang` включает более 20-ти расширений класса `Error` и примерно столько же расширений класса `Exception`. Однако большинство исключений из этого пакета являются наследниками класса `RuntimeException`, которые обрабатываются виртуальной машиной и не требуют обязательной обработки от программиста.

Пример: генерация исключений и ошибок

```
enum Crashes{
   Error, Exception,RuntimeException
}

class ExceptionsAndError{
 void genCrash(Crashes crash){
   switch(crash){
    case Error: throw new Error("Crash №1");
    case Exception: throw new Exception("Crash №2");
    case RuntimeException:
      throw new RuntimeException("Crash №3");
    default: return;
   }
}
}
class CrashDemo{
  public static void main(String[] args){
        ExceptionsAndError ee=new ExceptionsAndError();
        Crashes[] crashes=Crashes.values();
```

```
        for(Crashes crash : crashes) ee.genCrash(crash);
    }
}
```

Замечание к примеру: класс `CrashDemo` не будет компилироваться пока не будут выполнены требования к обработке исключения (`Exception`); к `Error` и к `RuntimeException` таких требований нет.

9.3.3 Класс Thread

Термином «threads» (нити) называют независимые последовательности команд, выполняемые на параллельно-конкурентной основе в рамках одной программы. Такая организация вычислений называется «multithreading», что переводится на русский как «многопоточность». Слово «поток» (stream) используется в смысле «поток данных», и чтобы избежать двусмысленности, будем переводить «thread» как «поток исполнения».

Java поддерживает многопоточность на уровне языка и основной библиотеки. Абстрактный класс Thread представляет модель потока исполнения. Единственный абстрактный метод run() класса представляет «рамку» для включения в поток последовательности команд, которые он будет исполнять. Для организации нового потока исполнения достаточно создать объект типа Thread, определив его метод run(), и вызвать у этого объекта метод start(). Виртуальная машина будет автоматически выполнять переключения между последовательностями команд всех зарегистрированных в ней потоков исполнения.

Пример: запуск нескольких потоков исполнения

```
class SimpleThreadDemo {
private static int i=1;

public static void main(String[] args){
Thread[] threads=new Thread[5];
while (i<6) {
        threads[i-1]=new Thread() {
          public void run(){
                String s=i+"-й пошел...";
                        System.out.println(s);
                try {
                while(true){
                        Thread.sleep(10);
                        System.out.println(s);
                }
                }catch(InterruptedException e){ return; }
        }
        };
```

```
            threads[i-1].start();
            try{
                    threads[i-1].join(10);
            }catch(InterruptedException e){}
            i++;

    }
    i=0;
    for(Thread t : threads) t.interrupt();

    }
    }
```

Замечания к примеру: обращения к методам `sleep(…)` и `join(…)` класса `Thread` заключаются в блоки `try-catch` потому, что они используют механизм исключений для получения сообщений от других потоков исполнения через объекты типа `InterruptedException`. Этот механизм запускается применением метода `interrupt()` к заданному экземпляру класса `Thread`.

9.3.4 Классы System и Runtime

Эти классы предоставляют программисту методы управления средой исполнения (виртуальной машиной) и связь через ее посредство с «внешним миром» - операционной системой. В классе System собраны статические компоненты виртуальной машины: стандартные потоки ввода-вывода, управление завершением работы, настройки системы. Runtime обеспечивает возможность запуска внешних процессов, мониторинг состояния памяти, вызов программы завершения (ShutdownHook).

Пример: применение методов классов System и Runtime

```
    class UseJVMDemo{
    private static void p(String s){
     System.out.println(s);
    }

    public static void main(String args[])throws Exception{
    p("Класс System:");
    p("Текущее время: "+System.currentTimeMillis());
    p("Настройки JVM:");
       System.getProperties().list(System.out);
    p("Системное окружение:");
    for(String key:System.getenv().keySet())
p(key+"="+System.getenv().get(key));
```

```
    p("Класс Runtime:");
    Runtime rt=Runtime.getRuntime();
    p("Получить ссылку на Runtime:"+rt.toString());
    p("Память: всего - "+rt.totalMemory()+" свободно -
"+rt.freeMemory()+" максимум - "+rt.maxMemory());
    p("Запустить внешний процесс:");
    rt.exec("firefox");
    p("Добавить \"Завершающий крюк\"");
    rt.addShutdownHook(new Thread(){
     public void run(){
     for(int i=0;i<10; i++){
          System.out.print(i+" ");
          try{
               Thread.sleep(100);
     }catch(InterruptedException e){}
     }
          System.out.println();
     }
    });
    p("Вызвать сборщика мусора...");
    rt.gc();
    p("Класс System: закончить выполнение с кодом завершения
13:");
    System.exit(13);
    }
```

9.3.5 Классы-строки

В библиотеку поддержки входят три класса, обеспечивающие манипуляции со строками символов. В классе String символы, составляющие строку, содержатся в массиве, длина которого не может изменяться. В этом классе собраны методы либо не изменяющие содержимое строки (такие как `char charAt(int pos)`), либо возвращающие новый объект (такие, как `String replace(CharSequence target, CharSequence replacement)`).

В классах `StringBuffer` и `StringBuilder` символы строки хранятся в структурах типов `Vector` и `ArrayList`, допускающих изменение числа элементов. Эти классы, полностью идентичные друг другу по наборам методов, содержат методы добавления, вставки и замены символов в исходном объекте, что невозможно с экземплярами класса `String`. Они не создают нового объекта, а изменяют исходный. Методы класса `StringBuffer` безопасны при применении в многопоточной среде (что накладно), класс `StringBuilder` предназначен для однопоточных приложений.

Пример.

```
class UseString{
private static void p(String s){
 System.out.println(s);
}

public static void main(String[] args){
p("String demo:");
String myname="Стесик Ольга Львовна";
p("Исходная строка: \""+myname +
 "\" длиной в "+myname.length()+" символов");
String converted=myname.toLowerCase();
p("В нижнем регистре: "+converted);
 converted=myname.toUpperCase();
p("В верхнем регистре: "+converted);
 converted=myname.replace(' ','+');
p("С заменой пробелов на +: "+converted);
 converted=myname.substring(7,13);
p("Часть 7-13 : "+converted);
 converted=myname.concat(" преподает информатику.");
p("Вместе с другой строкой : "+converted);

p("StringBuffer demo:");
/*
Объекты классов StringBuffer и StringBuilder изменяемы, с ни-
ми нет необходимости присваивать
результаты методов новым переменным
*/
StringBuffer buf=new StringBuffer(myname);
buf.insert(13,"свет ");
p("Вставим слово: " + buf);
buf.reverse();
p("Перевернем задом наперед: "+buf);

}
}
```

9.3.6 Классы-оболочки

Классами-оболочками называются классы, представляющие производный аналог встроенных типов данных и ключевого слова `void`: `Double`, `Float`, `Character`, `Integer`, `Byte`, `Short`, `Long`, `Boolean`, `Void`. Они, с одной стороны, представляют статические методы для преобразований объектов

соответствующих типов, например, из строки в число. С другой стороны, они необходимы там, где встроенные типы невозможны: в объектах классов, представляющих объединения других объектов (кроме массивов) и в механизме отражений.

Пример: массив-список объектов типа Double

```
import java.util.ArrayList;
import java.util.Random;

class UseWrappers{
public static void main(String[] args){
//Генератор случайных чисел
 Random rand=new Random();

//ArrayList - "неограниченный" массив объектов типа Double
  ArrayList<Double> al=new ArrayList<Double>();
//Заполним его приблизительно до 60-ти элементов порциями
// не больше 20 штук за раз
 int portion, size=0;
 while( size < 50 ){
  portion=rand.nextInt(20);
  System.out.println("В списке "+al.size()+
" объектов Double,");
  System.out.println("добавим еще "+portion+ " штук.");
  for( ; portion >0; portion--){
        double d=rand.nextDouble()*100 -100;
        al.add(new Double(d));
        size++;
  }
 }
  System.out.println("Всего в списке "+al.size()+
" объектов Double:");
  Double[] ad=new Double[al.size()];
  ad=al.toArray(ad);
  for(Double D : ad) System.out.print(D.toString()+" ");
  System.out.println();

}
```

9.3.7 Класс Math

Класс `Math` является библиотекой основных математических функций для вещественных аргументов. Содержит тригонометрические функции, квадратный корень, возведение в степень и функции округления, предусмотренные стандартом

IEEE-754. Версия `StrictMath` дает независящий от типа процессора результат (даже в ущерб точности). Константы `Math.PI` и `Math.E` представляют числа pi и e.

Пример: методы класса Math

```
class UseMath{
private static void p(String s){
 System.out.println(s);
}

public static void main(String[] args){
//Зададим число
double x=Math.PI+Math.E;
 p("x="+x);
//Вычислим некоторые функции от этого числа
//и напечатаем значения
 p("sin("+x+")="+Math.sin(x));
 p("cos("+x+")="+Math.cos(x));
 p("asin("+x+")="+Math.asin(x));
 p("tg("+x+")="+Math.tan(x));
 p(x+"**2.5="+Math.pow(x,2.5));
 p("exp("+x+")="+Math.exp(x));
 p("ln("+x+")="+Math.log(x));
 p("lg("+x+")="+Math.log10(x));
    }

}
```

9.3.8 Класс Class и подпакет java.lang.reflect

Объекты типа `Class` описывают классы и интерфейсы в программе, исполняемой виртуальной машиной. Объект класса `Class` возвращается методом `getClass()` класса `Object`. Методы класса `Class` обеспечивают механизм исследования классов. Классы из подпакета `java.lang.reflect` – `Constructor`, `Field`, `Method` – содержат представления составных частей класса – конструкторов, полей и методов. Статические методы `forName(..)` по заданному имени класса создают для выполняющегося приложения объект типа `Class` для указанного класса. Метод `newInstance()` создает из объекта типа `Class` новый экземпляр описываемого объектом `Class` класса.

Пример: метод Class.forName(…) и отражение загруженного класса

```
import java.lang.*;
import java.lang.reflect.*;
```

```
public class SimpleClassDemo {
public static void main(String[] args){
String[] names=
  {"java.lang.String","java.util.Random","java.lang.Integer"};
  for(String name : names ){
    System.out.println("Name="+name);
    try {
     Class c=Class.forName(name);
     Field[] fields=c.getFields();
     System.out.println("В классе "+name+
          " "+fields.length+" полей.");
     for(int j=0; j<fields.length; j++)
       System.out.println("Поле "+ j+": название "
            +fields[j].getName()+" тип "+fields[j].getType());
       Method[] methods=c.getMethods();
       System.out.println("В классе "+ name+
       " "+methods.length+ " методов.");
       for(int j=0; j<methods.length; j++){
            System.out.println("Метод "+ j+": название "
            +methods[j].getName()+"  "+methods[j].toString());
            if(methods[j].getName().equals("hashCode"))
            try{
                    Object obj=new Object();
                    System.out.println("hashCode="
                     +methods[j].invoke(c.newInstance(),obj));
            }catch(Exception e){e.printStackTrace();}
       }
      } catch(ClassNotFoundException e){
            System.out.println("Класс по имени "
            + name +" не найден.");}
      }
    }
    }
}
```

Замечание к примеру: попытка создать объект типа `Integer` из объекта `Class` для имени `java.lang.Integer` не приводит к успеху из-за отсутствия в классе `Integer` конструктора без параметров.

9.4 Интерфейсы пакета java.lang

В библиотеку поддержки включены интерфейсы, используемые в конструкциях языка или связанные с классами, поддерживающими их.

Эти основные интерфейсы включают объявление одного-двух методов, описывающих какое-то одно свойство: например, свойство сравнимости.

Среди них есть также интерфейсы-маркеры, не включающие ни одного метода: имплементация такого интерфейса является разрешением на совершение каких-то действий над объектами, совместимыми с маркером.

9.4.1 Интерфейс Runnable

Содержит единственный метод `void run()`. Объекты типа `Runnable` используются для создания новых экземпляров класса `Thread` – потоков исполнения.

Пример: графическое приложение «Панель с часами»

```
import java.awt.*;
import java.awt.event.*;
import java.util.Date;
import java.text.DateFormat;

class Watch extends Frame implements Runnable{
private static String date="Поехали!!!";
private TextField label=new TextField(date,60);

Watch(){
 super("My Watch");
 setLayout(new GridLayout(1,1));
 add(label);
 addWindowListener(new WindowAdapter(){
 public void windowClosing(WindowEvent we){
            System.exit(0);
      }
 });
setSize(300,60);
label.setBackground(Color.white);
label.setForeground(Color.blue);
label.setFont(new Font("courrier",Font.BOLD,12));
setVisible(true);
}

public void run(){
Date mydate=new Date();
long currentTime=mydate.getTime();
DateFormat df=
DateFormat.getDateTimeInstance(
DateFormat.FULL,DateFormat.FULL);
while(true){
 try{
```

```
     Thread.sleep(1000);
     }catch(InterruptedException ie){}
     currentTime+=1000l;
     mydate.setTime(currentTime);
     date=df.format(mydate);
     label.setText(date);
     }
    }

    public static void main(String[] args){
     Thread watch=new Thread(new Watch());
     watch.start();
    }
}
```

Замечание к примеру: класс `Watch` расширяет класс `java.awt.Frame` (окно с рамкой и тремя кнопками) и имплементирует интерфейс `Runnable`. Благодаря последнему, из экземпляра `Watch` можно создать поток исполнения, меняющий показания часов.

9.4.2 Интерфейс Comparable

Очень важный интерфейс, описывающий способность объектов к сравнению между собой. Содержит метод `int compareTo<T>(T other)`. Это «настраиваемый» метод – его можно приспособить для сравнения между собой объектов заданного, а не обобщенного –`Object` – типа.

9.4.3 Интерфейс Iterable

Этот интерфейс определяет способность объектов порождать механизм перебора: применяется к типам-контейнерам, таким как массивы, списки или наборы. Содержит один метод:

```
Iterator<T> iterator()
```

Тип `Iterator<T>` - настраиваемый интерфейс из пакета `java.util`; он описывает методы механизма перебора `boolean hasNext()` и `<T> next()`.

Пример: интерфейсы Iterable и Comparable

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;

class JavaLangIntrfsDemo{
static String[] names=new String[]
{"Хитрук","Иванов","Якименко","Петров","Сидоров","Алексеев"};
```

```
static int index=0;

static Object genObjects(){

Random rand=new Random();
int quality=rand.nextInt(2);
if(quality == 0 ) return new Object();
return names[index++];
}

public static void main(String[] args){
Iterable[] its=new Iterable[2];
 its[0]=new TreeSet();
 its[1]=new HashSet();
 while(index<6){
  Object next=genObjects();
  if( next instanceof Comparable )
      ((TreeSet) its[0]).add(next);
  else ((HashSet) its[1]).add(next);
 }
 for( Iterable it : its){
   Iterator iterator=it.iterator();
   while(iterator.hasNext())
       System.out.print(iterator.next().toString()+" ");
   System.out.println();
 }
}
}
```

Замечание к примеру: метод `genObjects` на случайной основе возвращает или одну строку из неупорядоченного списка фамилий, или новый объект класса `Object`. Метод `main(..)`, получая объект, проверяет его на совместимость с интерфейсом `Comparable`. Класс `String` имплементирует этот интерфейс, а класс `Object` – нет. Объекты `Comparable` помещаются в упорядочиваемый набор `TreeSet`, не совместимые с `Comparable` помещаются в не упорядочиваемый набор `HashSet`. Оба набора имплементируют интерфейс `Iterable`, что позволяет объединить их в массив и распечатать элементы обоих наборов в одном цикле. Фамилии выводятся в алфавитном порядке, каждый экземпляр класса `Object` представляется строкой вида «`имя_класса`»@«`хэшкод`».

9.4.4 Интерфейс Cloneable

Интерфейс-маркер, не содержащий ни одного метода. Имплементация этого интерфейса каким-либо классом показывает, что объекты этого класса можно «клонировать» - применять к ним метод `clone()` класса `Object`.

9.5 Задания

Для изучающих Java

1. Используя документацию и дополнительные источники, составьте детальное описание всех методов класса `java.lang.Object`.
2. Составьте таблицу всех классов типа `Exception` из пакета java.lang, укажите, в каких случаях они используются.
3. Выясните значение ключевого слова `synchronized`. Объясните, где оно применяется.
4. Выясните, какое действие выполняет метод `interrupt()` класса `Thread` и какое исключение описывает тип `InterruptionException`.

Для изучающих другой язык программирования

1. Исследуйте библиотеку поддержки изучаемого языка программирования – выясните ее название, устройство, состав и происхождение.
2. Составьте краткое описание всех (или основных) элементов библиотеки поддержки.
3. Напишите небольшие программы-примеры использования библиотеки поддержки в вашем языке программирования.

10 Литература

10.1 Для изучающих Java

1. Патрик Ноутон, Герберт Шилдт. Java™ 2. Наиболее полное руководство/Пер.с англ. – СПб.: БХВ-Петербург, 2008. - 1072 с.
2. Брюс Эккель, Философия Java, 5-е издание/Пер. с англ. – СПб.: Питер, 2016. – 1168 с.
3. В. Монахов, Язык программирования Java и среда NetBeans, 3-е издание – СПб.: БХВ-Петербург, 2012. – 704 с.

10.2 Для изучающих другие языки программирования

С: Б. Керниган, Д. Ритчи, Язык программирования С, 2-ое издание/Пер. с англ. – СПб.:Вильямс, 2015. – 304 с.

С++: Стивен Прата, Язык программирования С++. Лекции и упражнения, 6-ое издание/Пер. с англ. – СПб.:Вильямс, 2012. – 1248 с.

FORTRAN: С. Немнюгин, О. Стесик, Современный Фортран, - СПб.:БХВ, 2005. – 496 с.
Pascal:
JavaScript: Дэвид Флэнаган, JavaScript. Подробное руководство, 5-ое издание/Пер. с англ. – СПб.:Символ-Плюс,2008. – 992 с.
PHP: Д. Котеров, А. Костарев, PHP 5, 2-ое издание – СПб.:БХВ-Петербург, 2014. – 1104 с.
Lisp: Пол Грэм, ANSI Common Lisp, - СПб.:Символ-Плюс, 2012. – 448 с.
Python: Марк Лутц, Изучаем Python, 4-ое издание/Пер. с англ. – СПб.:Символ-Плюс, 2011. – 1280 с.