Оглавление

Глава 1. ВВЕДЕНИЕ В ООП И ФП

1.1 Базовые понятия ООП

ООП — методология программирования, основанная на функционировании программного продукта как результата взаимодействия совокупности объектов, каждый из которых является экземпляром конкретного класса.

Объект — именованная модель реальной сущности, обладающая конкретными значениями свойств и проявляющая свое поведение.

Класс — модель информационной сущности, представляющая универсальный тип данных, состоящая из набора полей данных и методов их обработки.

Свойства объекта и класса (универсальные):

- **Инкапсуляция** - объединение закрытых данных и открытых методов
- **Наследование** - заимствование функциональности базовых классов производными
- **Полиморфизм** - возможность использования объектов с одинаковыми интерфейсами при наследовании

1.2 Базовые понятия ФП

ФП – подход, предполагающий формирование функции как объекта для последующей передачи в метод или возвращения из него в качестве результата.

Примечания:

- Добавлено в Java для облегчения вычислений и обработки информации
- Не заменит другие подходы, т.к. многие последовательные процессы, (поведение программных моделей в реальном времени, игровые и др. программы, организующие взаимодействие ПК с человеком) не могут быть выражены в функциональном стиле
- Некоторые приемы функционального программирования могут с успехом использоваться и в традиционном программировании вместе с ООП

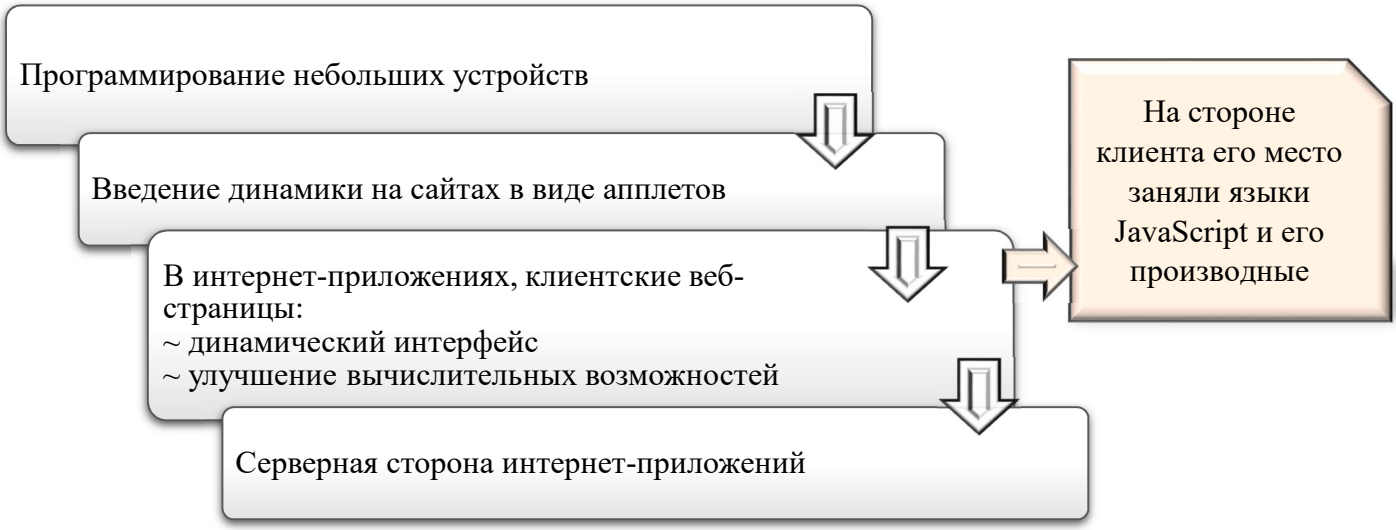
1.3 Базовые понятия Java

Стиль: ООП
Разработан: Sun Microsystems
Год создания: 1995
Синтаксический предок: C++
Системная библиотека:

- расширение базовых возможностей
- сетевые средства
- взаимодействие с БД
- многопоточность
- и мн.др…

Методы библиотеки вызываются JVM во время интерпретации

ПРИМЕНЕНИЕ JAVA



Размещение в памяти	
Объект	**Объектная ссылка*** **дескриптор объекта, т.к. содержит информацию о его классе*
в динамической памяти - куче данных (heap)	хранится в стеке (stack)
Преимущества	**Недостатки**
исключен непосредственный доступ к памяти	сложнее и менее эффективна работа с элементами массивов по сравнению с C++
наличие дескрипторов позволяет JVM выполнять проверку совместимости типов на фазе интерпретации кода, генерируя исключение в случае ошибки	* усовершенствован механизм работы с коллекциями, реализующими основные динамические структуры данных

C++	Java
Указатели	
+	Исключены, как требование безопасности: возможность работы с произвольными адресами памяти через безтиповые указатели позволяет игнорировать защиту памяти. Взамен используются ссылки.
Множественное наследование	
+	Заменяется на более понятные конструкции с применением интерфейсов
Концепция организации динамического распределения памяти	
Программное освобождение динамически выделенной памяти с помощью деструктора	Система автоматического освобождения памяти «сборщик мусора», выделенной с помощью оператора new, можно только рекомендовать системе освободить выделенную динамическую память
Перегрузка операторов	
+	-
Беззнаковые целые	
+	-
Прямое индексирование памяти (=> указатели)	
+	-

Конструкторы	
-	+
Деструкторы	
+	- **автоматическая сборка мусора*

КЛЮЧЕВЫЕ СЛОВА В JAVA:

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

***значение зависит от позиции в коде, используется вместо типа локальной переменной метода, для которой существует инициализатор, определяющий тип: var str = "sun"*

****новые*

**запрещены к использованию*

Литералы, не относящиеся к ключевым и зарезервированным словам:
null
true
false

1.4 Простое приложение

Программа передачи символьного сообщения на консоль.

```
// # 1 # простое линейное приложение # HelloTutorial.java
public class HelloTutorial {
        public static void main (String[] args) {
                System.out.println("tutorial->https://docs.oracle.com/javase/tutorial/");
        }
}
```

Здесь функция main() - начало выполнение любой программы Java*, является методом класса HelloTutorial. Такая простая структура класса приведет к проблемам при необходимости внесения изменений.

**автоматически вызывается интерпретатором при запуске программы*

ПРОБЛЕМА

Замена/изменение фразы для вывода на консоль потребует поиска мест ее использования по всему коду...

РЕШЕНИЕ

Для локализации изменений сообщение лучше хранить в отдельном методе или константе (а еще лучше — в файле) и при необходимости вызывать его

В следующем примере этот код будет переписан с использованием двух классов, реализованных на основе простейшего применения ООП:

```
/* # 2 # простое объектно-ориентированное приложение # FirstProgram.java */
package by.epam.learn.main;

public class FirstProgram {
        public static void main(String[] args) {
        // declaring and creating an object
                TutorialAction action = new TutorialAction();
        // calling a method that outputs a string
                action.printMessage("tutorial-> https://docs.oracle.com/javase/tutorial/");
        }
}
```

package by.epam.learn.main: принадлежность класса пакету (каталог на диске)
Класс FirstProgram определяет метод main(), по сути - контроллер приложения
Метод main():

- **public** член класса - может быть виден и доступен любому классу
- **static** - может быть использован при работе с классом, без создания его объекта
- **принимаемый параметр** - аргументы командной строки String[]args (массив строк)
- **void** – метод не возвращает значение

Тело содержит:

- **объявление объекта:** TutorialAction action = new TutorialAction();
- **вызов его метода:** action.printMessage("tutorial-> https://docs.oracle.com/javase/tutorial/");

```
/* # 3 # простой класс # TutorialAction.java */
class TutorialAction {
        void printMessage(String msg) {
        // method definition
        // output string
                System.out.println(msg)
        }
}
```

class TutorialAction: прототип объекта созданного в контроллере
Метод printMessage:

- **доступ не указан => default,** внутри текущего пакета
- **метод объекта (non-static)** - может быть вызван только на объекте
- **принимаемый параметр** - String msg объект типа строка

Тело содержит:

- фактический **вывод строки методом System.out.println**:
  - System - класс встроенного пакета java.lang (подключается автоматически)
  - out - статическое поле-объект класса System
  - println() (ln - переход к новой строке) - метод объекта out

Приведенную программу необходимо поместить в файл FirstProgram.java, имя которого должно совпадать с именем public-класса. Для приложения, состоящего из двух классов, наличие пакетов не является необходимостью. Однако даже при отсутствии слова package классы будут отнесены к пакету по умолчанию (unnamed), размещенному в корне проекта.

Классы из примеров 2 и 3 могут сохраняться как в одном файле, так и в двух файлах FirstProgram.java и TutorialAction.java, но хранение классов в отдельных файлах облегчает восприятие концепции приложения в целом:

```
* # 4 # простое объектно-ориентированное приложение # FirstProgram.java */
package by.epam.learn.main;
import by.epam.learn.action.TutorialAction; // import a class from another package

public class FirstProgram {
	public static void main(String[] args) {
		TutorialAction action = new TutorialAction();
		action.printMessage("tutorial-> https://docs.oracle.com/javase/tutorial/");
	}
}
```

```
*/ # 5 # простой класс # TutorialAction.java
package by.epam.learn.action;

public class TutorialAction {
	public void printMessage(String msg) {
		System.out.println(msg);
	}
}
```

1.5 Установка JDK и IDE

Установка Java с официального сайта производителя:
https://www.oracle.com/java/technologies/javase-downloads.html
При инсталляции рекомендуется указывать для размещения корневой каталог. Если JDK установлена в директории (для Windows**) c:\Java\jdk***[version]*, то каталог, который компилятор Java будет рассматривать как корневой для иерархии пакетов, можно вручную задавать с помощью переменной среды окружения в виде:
CLASSPATH=.;c:\Java\jdk*[version]*\.
Переменной задано еще одно значение **«.»** для использования текущей директории, например, **c:\workspace** в качестве рабочей для хранения своих собственных приложений.
Чтобы можно было вызывать сам компилятор и другие исполняемые программы, переменную **PATH** нужно проинициализировать в виде **PATH=c:\Java\jdk***[version]***bin**.
Этот путь указывает на месторасположение файлов **javac.exe** и **java.exe**.

JDK (Java Development Kit) — полный набор для разработки и запуска приложений:

- Компилятор
- Утилиты
- Исполнительная система JRE
- Библиотека
- Документация

JRE (Java Runtime Environment) — минимальный набор для исполнения приложений, включающий JVM, но без средств разработки.

JVM (Java Virtual Machine) — базовая часть исполняющей системы Java, которая интерпретирует байт-код Java, скомпилированный из исходного текста Java-программы для конкретной операционной системы.
Однако при одновременном использовании нескольких различных версий компилятора и различных библиотек применение переменных среды окружения начинает мешать эффективной работе, так как при выполнении приложения поиск класса осуществляется независимо от версии. Когда виртуальная машина обнаруживает класс с подходящим именем, она его и подгружает.
Такая ситуация предрасполагает к ошибкам, порой трудноопределимым. Поэтому переменные окружения начинающим программистам лучше не определять вовсе.

Способы компиляции и запуска приложений:

- Командная строка
- IDE (IntelliJ IDEA, Eclipse, NetBeans etc.)

IntelliJ IDEA— интегрированная среда разработки ПО для Java, компаниеи JetBrains.
Загрузить IntelliJ IDEA можно на официальном сайте:
https://www.jetbrains.com/idea/.
Регистрационная форма для установки полной лицензионной версии студентам:
https://www.jetbrains.com/shop/eform/students.

Eclipse — сообщество открытого кода, или open source, чьи проекты сфокусированы на создании открытой платформы для разработки, развертывания, управления приложениями с использованием различных фреймворков, инструментов и сред исполнения.
Загрузить продукты Eclipse можно на официальном сайте:
http://eclipse.org/

1.6 Компиляция и запуск приложения

Вызов строчного компилятора из корневого каталога:

```
javac by/epam/learn/action/TutorialAction.java
javac by/epam/learn/main/FirstProgram.java
```

** при успешной компиляции создаются файлы FirstProgram.class TutorialAction.class*

Запуск байт-кода с помощью интерпретатора Java:

```
java by.epam.learn.main.FirstProgram
```

** последовательность строк, разделенных пробелами, значения которых присваиваются объектам массива String[]args*

Обработка аргументов командной строки метода main():

- args[0] = значение первой строки после имен компилятора и приложения.
- args.length - количество аргументов

** константное поле класса массива, не м.б. изменено на протяжении всего жизненного цикла*

/* # 6 # вывод аргументов командной строки в консоль # PrintArgumentMain.java */

```
package by.epam.learn.arg;

public class PrintArgumentMain {
        public static void main(String[] args){
                for (String str : args) {
                        System.out.printf("Argument--> %s%n", str);
                }
        }
}
```

Аналог, классический цикл for:

```
for (int i = 0; i < args.length; i++) {
System.out.println("Argument--> " + args[i]);
}
```

Тело main() содержит:

- **цикл for** для неиндексируемого перебора всех элементов
- метод **форматированного вывода printf()**

Запуск с помощью командной строки:

```
java by.epam.learn.arg.PrintArgumentMain 2020 Oracle "Java SE"
```

Вывод на консоль:

```
Argument--> 2020
Argument--> Oracle
Argument--> Java SE
```

1.7 Основы классов и объектов

Классы в языке Java объединяют:

- поля класса
- методы
- конструкторы
- логические блоки
- внутренние классы

C++	Java
Объявление функций:	
возможно вне класса/методов класса	все функции определяются только внутри классов и называются **методами**
Область действия спецификаторов доступа (public, private, protected)	
на участок до следующего спецификатора	только на те объявления полей, методов и классов, перед которыми указаны
Уровень доступа к элементам с модификатор по умолчанию	
private	текущий пакет

Общий вид объявления класса:

[*specificators*] class *ClassName* [extends *SuperClass*] [implements *list_interfaces*] {
    /* implementation */ }

Спецификатор доступа к классу [*specificators*] может быть:

- **public** - доступен в данном пакете и вне пакета
- **protected** – доступен в текущем пакете и во всех подклассах
- **default** (не задан, friendly или private-package) - доступен только в текущем пакете

Другие спецификаторы класса:

- **final** - не может иметь подклассов/наследников
- **abstract** - может содержать абстрактные нереализованные методы, объект такого класса создать нельзя

[extends *SuperClass*] - класс наследует все свойства и методы суперкласса (базового класса), который указан после ключевого слова extends.

[implements *list_interfaces*] - может включать множество интерфейсов*, перечисленных через запятую после ключевого слова implements.

** Интерфейсы относительно похожи на абстрактные классы, содержащие только статические константы и не имеющие конструкторов, но имеют целый ряд серьезных архитектурных различий.*

В качестве примера с нарушением инкапсуляции можно рассмотреть класс Coin в приложении по обработке монет.

// # 7 # простой пример класса носителя информации # Coin.java

```
package by.epam.learn.bean;

public class Coin {

	public double diameter; // encapsulation violation
	private double weight;  // correct encapsulation

	public double getDiameter() {
		return diameter;
	}

	public void setDiameter(double value) {
		if (value > 0) {
			diameter = value;
		} else {
			diameter = 0.01; // default value
		}
	}

	public double takeWeight() { // incorrect method name
		return weight;
	}

	public void setWeight(double value) {
		weight = value;
	}
}
```

**в некоторых ситуациях замена некорректного значения на значение по умолчанию может привести к более грубым ошибкам в дальнейшем, поэтому часто вместо замены производится генерация исключения.*

Класс Coin содержит два поля:

- **diameter: public**, объявлено с нарушением «тугой» инкапсуляции, т.е. доступно непосредственно через объект класса Coin, следствием чего может быть нарушение корректности информации, как это показано в примере #8
- **weight: private**, изменение доступно только методом setWeight(double value)

// # 8 # демонстрация последствий нарушения инкапсуляции # CoinMain.java

```
package by.epam.learn.main;

import by.epam.learn.bean.Coin;

public class CoinMain {

	public static void main(String[] args) {
		Coin coin = new Coin();
		coin.diameter = -0.12; // incorrect: direct access => ob.setWeight(100);
	// coin.weight = -150;  field is not available: compile error
	}
}
```

ПРОБЛЕМА	РЕШЕНИЕ
Возможность прямого присваивания значения полю с помощью ссылки на объект, в обход проверки корректности данных	Поле diameter класса Coin объявить с модификатором доступа private => попытки прямого присваивания приведут к ошибке компиляции

// # 9 # «туго» инкапсулированный класс (Java Bean) # Coin.java

```
package by.epam.learn.bean;

public class Coin {

	private double diameter;
	private double weight;

	public double getDiameter() {
		return diameter;
	}

	public void setDiameter(double value) {
		if(value > 0) {
			diameter = value;
		} else {
			System.out.println("Negative diameter!");
		}
	}

	public double getWeight() { // correct name
		return weight;
	}

	public void setWeight(double value) {
		weight = value;
	}
}
```

Проверка корректности входящей извне информации позволяет уведомить о нарушении инициализации объекта

/* # 10 # создание объекта, доступ к полям и методам объекта # CompareCoin.java # CoinMain.java */

```
package by.epam.learn.action;

import by.epam.learn.bean.Coin;

public class CompareCoin {

	public void compareDiameter(Coin first, Coin second) {
		double delta = first.getDiameter() - second.getDiameter();
		if (delta > 0) {
			System.out.println("The first coin is more than the second for " + delta);
		} else if (delta == 0) {
			System.out.println("Coins have the same diameter");
		} else {
			System.out.println(
			"The second coin is more than the first on " + -delta);
		}
	}
}
```

```
package by.epam.learn.main;

import by.epam.learn.bean.Coin;
import by.epam.learn.action.CompareCoin;
public class CoinMain {

	public static void main(String[] args) {
		Coin coin1 = new Coin();
		coin1.setDiameter(-0.11); // error message
		coin1.setDiameter(0.12); // correct
		coin1.setWeight(150);

		Coin coin2 = new Coin();
		coin2.setDiameter(0.21);
		coin2.setWeight(170);

		CompareCoin compare = new CompareCoin();
		compare.compareDiameter(coin1, coin2);
	}
}
```

** доступ к public-методам объекта класса осуществляется только после создания объекта данного класса*

Вывод на консоль:

```
Negative diameter!
The second coin is more than the first on 0.09
```

Этапы создания объекта класса:

** обычно объединяют в одно объявление:*
Coin coin = new Coin();

- **объявление ссылки** на объект класса
  Coin coin; *// declaring an object reference*
- **создание экземпляра** объекта с помощью оператора **new**
  coin = new Coin(); *// object instantiation*
  - оператор new вызывает конструктор, в данном примере - по умолчанию без параметров, но конструктор может принимать аргументы для инициализаци, при соответствующем объявлении в классе

- операция присваивания (=) для объектов означает, что две ссылки будут указывать на один и тот же участок памяти

ПРОБЛЕМА

compareDiameter(Coin first, Coin second) выполняет два слишком различных действия, которые следует разделить:

- выполняет сравнение
- печатает отчет

РЕШЕНИЕ

изменить возвращаемое значение метода на int и оставить в нем только вычисления, а формирование отчета следует поместить в другой метод другого класса

/* # 11 # метод сравнения экземпляров по одному полю # */

```
public int compareDiameter(Coin first, Coin second) {
        int result = 0;
        double delta = first.getDiameter() - second.getDiameter();

        if (delta > 0) {
                result = 1;
        } else if (delta < 0) {
                result = -1;
        }

        return result;
}
```

1.8 Объектные ссылки

Java работает не с объектами, а со ссылками на объекты, размещаемыми в динамической памяти с помощью оператора new => операции сравнения ссылок - сравнивают адреса. Сравнение объектов на эквивалентность по значению – используя специальные методы, например, equals(Object ob). Этот метод наследуется каждый классом от суперкласса Object, (в корне дерева иерархии всех классов) и должен переопределяться в подклассе для определения эквивалентности содержимого двух объектов этого класса.

/* # 12 # сравнение ссылок и объектов # ComparisonString.java */

```
package by.epam.learn.main;

public class ComparisonString {

        public static void main(String[ ] args) {
                String str1, str2;

                str1 = "Java";
                str2 = str1; // variable refers to the same string
                System.out.println("comparison of references " + (str1 == str2)); // true

                str2 = new String("Java"); // is equivalent to str2 = new String(str1);
                System.out.println("comparison of references "+ (str1 == str2)); // false
                System.out.println("comparison of values " + str1.equals(str2)); // true
        }
}
```

обе ссылки ссылаются на 1 объект

при сравнении ссылок, оператор «==» возвращает true только если они ссылаются на 1 объект (один адрес)

конструктор new String(str1) - создает новый объект в другом участке памяти, инициализируемый значением из str1

сравнение значений объектов методом equals, переопределенном в классе String

Если в процессе разработки возникает необходимость в сравнении по значению объектов классов, созданных программистом, для этого следует переопределить в данном классе метод equals(Object ob) в соответствии с теми критериями сравнения, которые существуют для объектов данного типа или по стандартным правилам, заданным в документации.

1.9 Консольный ввод\вывод

Консоль - способ взаимодействия с программой посредством потоков ввода\вывода.
В Java взаимодействие с консолью обеспечивает статические поля класса System:

- in
- out
- err

Ввод данных

Вывод данных

// # 13 # чтение символа из потока System.in # ReadCharMain.java 24

```
package by.epam.learn.console;

import java.io.IOException;

public class ReadCharMain {

	public static void main(String[] args) {
		int x;
		try {
			x = System.in.read();
			char c = (char)x;
			System.out.println("Character Code: " + c + " =" + x);
		} catch (IOException e) {
			System.err.println("i\o error " + e);
		}
	 }
}
```

чтение числового кода из консоли

перевод числового кода в символ путем приведения типов

В блоке **try-catch -** обработка исключения **IOException**, возникающего в операциях ввода/вывода и других взаимодействиях с внешними источниками данных**:**

- блок **try**: выполняется, если ошибок не возникает
- блок **catch**: перехватывает выполнение при генерации исключения

С помощью объекта класса **Scanner** ввод информации осуществляется посредством чтения строки из консоли. Такой объект может соединяться практически с любым потоком/источником информации:

- строкой
- файлом
- сокетом
- адресом в интернете
- с любым объектом, из которого можно получить ссылку на поток ввода

// # 14 # чтение строки из консоли # ScannerMain.java

```
package by.epam.learn.console;

import java.util.Scanner;

public class ScannerMain {

	public static void main(String[] args) {
		System.out.println(
			"Enter name and press <Enter> & number and press <Enter>:");
		Scanner scan = new Scanner(System.in);
		String name = scan.next();
		System.out.println("hello, " + name);
```

создание объекта класса Scanner, с передачей в конструктор входящего консольного потока

создание переменной типа String и присвоение ей, запрошенной строки до первого пробела

```
            int num = scan.nextInt();
            System.out.println("number= " + num);

            scan.close();
        }
}
```

создание переменной типа int и присвоение ей, запрошенного целочисленного значения

Закрытие ресурса

Вывод на консоль:

```
Enter name and press <Enter> & number and press <Enter>:
ostap
hello, ostap
777
Number= 777
```

Класс **Scanner** объявляет ряд методов для ввода:

- next()
- nextLine()
- nextInt()
- nextDouble() и др.

1.10 Base code conventions

- **Имена классов, полей, методов**:
  - цельные слова, исключив сокращения, кроме общепринятых
  - по возможности без предлогов и очевидных связующих слов
  - аббревиатуры допустимы только когда они очевидны
  - если сокращения необходимо:
    - начало важнее конца
    - согласные важнее гласных
- **Имя класса:**
  - всегда пишется с большой буквы: Coin, Developer
  - если состоит из двух и более слов, то используется CamelCase:
    - AncientCoin, FrontendDeveloper.
- **Имя метода:**
  - всегда пишется с маленькой буквы:
    - perform(), execute()
  - если состоит из двух и более слов, то используется CamelCase:
    - performTask(), executeBaseAction().
- **Имя поля класса**, **локальной переменной и параметра метода**:
  - всегда пишутся с маленькой буквы:
    - weight, price.
  - если из двух и более слов, то используется CamelCase:
    - priceTicket, typeProject
- **Константы и перечисления:**
  - пишутся в верхнем регистре:
    - DISCOUNT, MAX_RANGE.
- **Имена пакетов**:
  - пишутся с маленькой буквы.
  - сокращения допустимы только, если имя пакета более 10 символов.
  - использование цифр и других символов нежелательно

1.11 Вопросы к главе 1

1. **Что имеется в виду, когда говорится:**
   a. **Java-язык программирования:**
   высокоуровневый, статически-типизированный и объектно-ориентированный формальный язык, предназначенный для записи компьютерных программ и определяющий набор лексических, синтаксических и семантических правил.
   b. **Java-платформа:**
   набор компонентов для разработки и запуска java приложений
2. **Расшифровать аббревиатуры, описать и показать физическое расположение:**
   a. **JVM:**
   Java Virtual Machine. Виртуальная машина java - исполняет байт код, созданный компилятором java (javac) из исходного кода
   …\Java\jdk1.version\jre\bin\server\jvm.dll
   b. **JDK:**
   Java Development Kit. Комплект разработчика Java, компонент для разработки Java приложений.
   …\Java\jdk1.version
   c. **JRE:**
   Java Runtime Environment. Среда выполнения Java - минимальная реализация, необходимая для запуска java приложений, без компилятора и среды разработки.
   …\Java\jdk1.version\jre
3. **JVM-JDK-JRE. Кто кого включает и как взаимодействуют:**
   JDK включает в себя java компилятор (javac) и JRE, который включает ключевой компонент платформы – JVM.
4. **Как связаны имя Java-файла и классы, которые в этом файле объявляются?**
   Один публичный класс должен соответствовать по имени одному файлу с расширением java. Это ограничение введено для поиска классов компилятором при запуске программы.
   Вложенные классы данного класса с другими модификаторами доступа и именами и статические классы с другими именами могут находиться в данном файле, при компиляции для каждого класса будет создан файл с расширением.class. для вложенных классов будет создан файл типа:
   ОсновнойКласс$ВложенныйКласс.class. (ключевое разделение с помощью знака $)
5. **Как скомпилировать и запустить класс, используя командную строку?**
   a. **Компиляция:**
   <u>Общий принцип: javac [ключи] [исходники]</u>
   Варианты запуска из директории с файлом класса:*
   - *javac MyClass.java - для конкретного класса*
   - *javac *.java - для всех классов в каталоге*

   **для запуска из любой директории - указать полный путь к файлам .java*
   По умолчанию скомпилированные файлы создаются в том же каталоге, где находятся файлы .java
   <u>Основные ключи:</u>
   - *-d позволяет указать компилятору, куда записать скомпилированные файлы сохраняя структуру пакета*
     **например файлы .java находятся в пакете com.company, в папке src. чтобы скомпилировать файлы .class в директории out, нам необходимо из корня проекта прописать javac -d out src\com\company*.java, итоговые файлы скомпилируются в out\com\company\MyClasses.class*
   - *-sourcepath указывает компилятору каталоги, в которых он должен иерархию исходных классов. пример : чтобы скомпилировать все классы из*

папки src пакета com.company в папку out вместе с их иерархией необходимо прописать следующее:
javac - d out -sourcepath src src\com\company.java*

b. Запуск приложения

Общий вид запуска класса: java [ключи] MyClass [аргументы]
**запускаем класс с main методом.*

6. Что такое classpath?

Classpath – это переменная окружения, которая сообщает JVM, где искать пользовательские и системные классы. Это может быть путь к директории, где хранятся скомпилированные классы, или путь к jar-файлам.
В Windows:

- *В командной строке можно установить classpath*:*
  set CLASSPATH=C:\path\to\classes; C:\path\to\jar\file.jar
- *при запуске java-приложения с помощью опции -cp или –classpath*:*
  java -cp /path/to/classes:/path/to/jar/file.jar com.example.MainClass

**действует только в рамках текущей сессии командной строки*
В большинстве современных IDE (например, IntelliJ IDEA, Eclipse) установка Classpath происходит автоматически при добавлении зависимостей

Зачем в переменных среды окружения прописывать пути к установленному JDK?

PATH:
Это позволяет нам запускать основные файлы java в любом месте из командной строки. Если этого не сделать, то, например, вызов компилятора javac нам пришлось бы делать из папки с установленной jdk.
как это работает: при вводе названия javac ОС проходит по всем путям и ищет исполняемый файл, исполняемые файлы в jdk находятся в папке bin. по этому в PATH мы прописываем путь вплоть до папки bin
JAVA_HOME:
используется приложениями на java для определения места установки JRE

7. Если в classpath есть две одинаковые библиотеки (или разные версии одной библиотеки), объект класса из какой библиотеки создастся?

Если в classpath есть несколько одинаковых библиотек, создастся класс из библиотеки, объявленной первой.

8. Объяснить различия между терминами «объект» и «ссылка на объект».

a. *Объект - экземпляр класса, находится в динамической памяти heap.*
b. *Ссылка на объект - адрес ячейки в памяти, указывающий расположение объекта, хранится в стеке (stack)*

9. Какие области памяти использует Java для размещения простых типов, объектов, ссылок, констант, методов, пул строк и т. д

a. *Java Heap (куча) используется Java Runtime для выделения памяти под объекты и JRE классы. Создание нового объекта также происходит в куче. Здесь работает сборщик мусора: освобождает память путем удаления объектов, на которые нет каких-либо ссылок. Куча используется всеми частями приложения => любой объект, созданный в куче, имеет глобальный доступ. Память в куче живет с самого начала до конца работы программы. -Xms и -Xmx опции JVM – для определения начального и максимального размера памяти в куче. Если память кучи заполнена, то бросается исключение java.lang.OutOfMemoryError: Java Heap Space*
b. *Stack память в Java работает по схеме LIFO => из-за простоты распределения памяти, стековая память работает намного быстрее кучи. Всякий раз, когда вызывается метод, в памяти стека создается новый блок, который содержит примитивы и ссылки на другие объекты в*

методе. Как только метод заканчивает работу, блок также перестает использоваться, предоставляя доступ для следующего метода. Размер стековой памяти намного меньше объема памяти в куче. Стек используется только одним потоком исполнения программы, не доступен для других потоков. Стековая память существует лишь какое-то время работы программы. Определить размер стека можно с помощью опции –Xss. Если память стека полностью занята, то Java Runtime бросает java.lang.StackOverflowError. Размер стека намного меньше памяти в куче.

10. **Почему метод main() объявлен как public static void?**
    Т.к. он является точкой входа в приложение:
    - *public – доступен для JVM (глобальный доступ)*
    - *static – не требует создания объекта для вызова (на момент запуска еще не создано ни одного объекта)*
    - *void – являясь точкой запуска по природе своего функционала не должен ничего возвращать*
11. **Возможно ли в сигнатуре метода main() поменять местами слова static и void?**
    Нет
12. **Будет ли вызван метод main() при запуске приложения, если слова static или public отсутствуют?**
    Нет
13. **Классы какого пакета импортируются в приложение автоматически?**
    java.lang

1.12 Задания к главе 1

Вариант А
1. Приветствовать любого пользователя при вводе его имени через командную строку.
2. Отобразить в окне консоли аргументы командной строки в обратном порядке.
3. Вывести заданное количество случайных чисел с переходом и без перехода на новую строку.
4. Ввести пароль из командной строки и сравнить его со строкой-образцом.
5. Ввести целые числа как аргументы командной строки, подсчитать их суммы и произведения. Вывести результат на консоль.
6. Вывести фамилию разработчика, дату и время получения задания, а также дату и время сдачи задания.

Вариант В
Ввести с консоли n целых чисел. На консоль вывести:
1. Четные и нечетные числа.
2. Наибольшее и наименьшее число.
3. Числа, которые делятся на 3 или на 9.
4. Числа, которые делятся на 5 и на 7.
5. Все трехзначные числа, в десятичной записи которых нет одинаковых цифр.
6. Простые числа.
7. Отсортированные числа в порядке возрастания и убывания.
8. Числа в порядке убывания частоты встречаемости чисел.
9. «Счастливые» числа.
10. Числа-палиндромы, значения которых в прямом и обратном порядке совпадают.
11. Элементы, которые равны полусумме соседних элементов.

1.13 Тестовые задания к главе 1

Вопрос 1.1.
Какие объявления параметров корректны для метода public static void main? (выбрать два)
a) String args []
b) String [] args
c) Strings args []
d) String args

Вопрос 1.2.
Какой из предложенных пакетов содержит class System? (выбрать один)
a) java.io
b) java.base
c) java.util
d) java.lang

Вопрос 1.3.
Какая команда выполнит компиляцию Java приложения First.java? (выбрать один)
a) javac First
b) java First.class
c) javac First.java
d) java First.java
e) java First

Вопрос 1.4.
Какие слова являются ключевыми словами Java? (выбрать два)
a) classpath
b) for
c) main
d) out
e) void

Вопрос 1.5.
Дан код:

```
public class Main {
        public static void main(String[] args) {
                for(int x = 0; x < 1; x++)
                System.out.print(x);
                System.out.print(x);
        }
}
```

Что будет в результате компиляции и запуска? (выбрать один)
a) 01
b) 00
c) 11
d) compilation fails
e) runtime error

Вопрос 1.6.

Дан код:

Каков будет вывод в консоль, если код запускается из командной строки java P R I V E T ? (выбрать один)

```
public class P {
        public static void main(String[] s) {
                System.out.print(s[1] + s[3] + s[2] + s[1]);
        }
 }
```

a) PIRP

b) IEVI

c) RVIR

d) compilation fails

e) runtime error