1. **История языка программирования Java. Достоинства и недостатки языка. Основные особенности. Жизненный цикл программы на Java.**

90-е, Джемс Гослинг, бытовые приборы, программисты для по и серверов, в 10 выкуплена Oracle.   
+: ООП, Авто определение памяти, популярность, многопоточность, простой синтаксис, независимость от платформы.

-: Маленькая скорость, авто распределение памяти, коммерческое использование

Особенности: не только язык, но и набор инстрементов. JDK, JRE, IDE

Цикл: .java -> .class -> JVM

1. **Типы данных Java. Особенности использования.**

Примитивные – значение, присваивание, строго ограничены

Ссылочные – ссылку, конструктора,

Примит: целочисленные, с плавающей точкой, булиан, чар

Ссылочные: массивы, коллекции

Область видимости: в одном блоке кода,

1. **Классы и объекты. Структура класса. Конструкторы класса. Блоки инициализации.**

Класс – шаблонная конструкция, благодаря которой в программе можно создать объект с определенными свойствами и поведением.

Объект – реализация Класса с определенными параметрами

Класс- чертёж, объект – воплощение

Структура:

Поля – переменные

Методы – блок кода

Конструктор:

Конструктор – это метод которые запускается один раз при создании объекта, позволяет присвоить значения переменным и исполнить какой-либо код

Блоки инициализации:

Если мы создали поля и не присвоили им значения, то мы можем после это в фигурных скобках присвоить этим полям значения

1. **Ключевое слово this, super, static (особенности инициализации), final.**

This: указывает на наш текущий класс

public String name;

public void TestClass(String name){

this.name = name

}

Super: используем, когда нужно вызвать конструктор родительского класса или когда нам нужно обратиться к полю\методу родительского класса

Static: делает поле\метод статичным. Статичные переменные не относятся к конкретному объекту. Методы могут взаимодействовать только со статичными полями. И методы, и поля можно вызвать через имя класса, не создавая при этом объект.

Final: подчёркивает завершённость. Поля с этой пометкой становятся константами и их нельзя изменить в будущем. Методы нельзя переопределять в дочерних классах. А классы нельзя сделать дочерними.

1. **Инкапсуляция. Реализация инкапсуляции в языке.**

Инкапсуляция объединяет данные и код в единую оболочку и позволяет ими манипулировать(классы). Защита.

Public – доступна везде

Privat – доступна только в текущем классе

Package-Privat – доступна только в текущем пакете

Protected – в текущем пакете или наследникам

Удобства:

1 Мы скрываем все не нужные пользователю данные

2 Изменения в методе никак не отражаются на пользователе

3 Пользователю не нужно вникать в тонкости работы программы, а достаточно вызвать нужный метод

1. **Наследование классов. Модификаторы доступа при наследовании. Преобразование классов при наследовании.**

Наследование – создание одного класса на основе другого. Родительский и дочерний. Extends. Super.

Public и Private

Upcasting, Downcasting. Первое доступно всегда доч-> род. Второе - род-> доч - не всегда доступно. Должен быть того класса, к которому преобразуется.

1. **Полиморфизм. Реализация полиморфизма в языке.**

Полиморфизм – представление объектов с одинаковой спецификацией иметь разную реализацию.

Представим что у нас есть метод GetArea()

Перегрузка: два одиковых метода, но разные параметры

Переопределение: переписывание метода(@Override)

1. **Абстрактные классы и методы. Интерфейсы. Назначение и использование абстрактных классов и интерфейсов. Члены интерфейсов и абстрактных классов.**

Абстрактный метод – это метод, у которого нет реализации, наша заготовка для будущего метода.

Абстрактный класс будет таковым являться если хотя бы один метод в нем является абстрактным. Нужно дописать abstract.

Такие классы нельзя использовать в готовом виде, так как абстрактные методы еще не реализованы. В производных классах мы обязаны реализовать все абстрактные методы.

Интерфейсы похожи на абстрактный класс, но есть некоторые отличия. Интерфейс так же содержит в себе абстрактные методы, но в отличии от абстрактный классов мы не обязаны реализовывать их все.

Абстрактные классы могут в себе содержать как поля, так и методы, в отличии от интерфейсов (public static final)

Абстрактные классы описывают более маленькую группу веще в отличии от интерфейса.

1. **Вложенные классы. Особенности использования. Классификация вложенных классов. Примеры.**

Вложенные классы - это классы, которые описаны в теле другого класса

class Test1() {

…

class Test2() {

…

}

}

Особенности:

* Test2 имеет доступ ко всем переменным тест1
* Экземпляры класса тест2 могут создаваться только в рамках класса тест1
* В классе тест2 нет модификаторов доступа.
* Живёт пока есть хотя бы одна ссылка на тест2

Вложенные классы бывают так же локальными. Их главное отличие в том, что локальный не ссылается на поля и методы главного класса

Анонимный класс- это класс без имени и реализуется сразу при создании объекта.

Car car = new Car(){

void wrum(){

System.out.println(“Врум врум врум”);

}

}

Статический вложенный класс – это вложенный класс объявленный с модификатором static. Он может взаимодействовать только со статичными полями и методами.

class Car(){

public static String noWrum = “no wruuum :(”

public static class Electrocar {

void wrum(){

System.out.println(noWrum);

}

}

}

1. **Коллекции объектов в Java. Интерфейсы и реализации коллекций. Итераторы.**

Коллекция представляет собой объект хранящий другие объекты.

Самым простым примерм выступает массив, но у него много ограничений, например он фиксированной длины.

Существуют два основных интерфейса:

* Collection – хранит объекты в виде списка. К любому объекту можно получить доступ и проводить с ним манипуляции.
* Map – хранит значения в виде пары «ключ-значение»

3 различных реализации коллекции:

* List – список объектов. Мы можем добавлять, удалять, изменять и извлекать их.
* Set – практически тоже самое что и лист, но может хранить в себе только единственную копию объекта.
* Map – объекты храниться в виде пар «ключ - значение»

Итератор – это простыми словами переборщик. Он создан для прохождения по коллекции. Основные методы итератора:

* hasNext() – возвращает значения тру и фолс в зависимости от того есть ли следующий элемент в коллекции или нет
* next() – переключение на следующий элемент коллекции
* remove() – удаление последнего элемента полученного от некст

1. **Исключения (exceptions) в языке Java. Порождение и перехват исключений. Список исключений в объявлении метода. Источники исключений. Иерархия исключений. Исключительные ситуации и наследование.**

Исключения предназначены для того чтобы при возникновении ошибки программа не завершалась, а выдавала ошибку или предупреждение.

Существуют очевидные и программные исключения.

Программные исключения срабатывают при компилировании программы и выдаёт ошибку.

Очевидно заданные – это вручную прописанные некоторые исключения. Throw.

При возникновении исключения программа останавливается(если мы не отлавливаем ошибку). Все последующие действия после ошибки не выполняются. Программа поочерёдно завершает все запущенные методы и проверяет не отлавливаем мы их. Если ошибка произошла в одном из потоков, то завершается только этот поток.

Чтобы отловить исключение нужно воспользоваться try-catch-finally.

Try{

//код с предполагаемым возникновение ошибки

}catch(названиеОшибки объектОшибки){

//делаем что-то если ошибка

}finally{

//в этом месте код исполняется в любом случае не зависимо от того была ли //ошибка или нет

}

Базовым классом для исключений является Throwable. От него наследуется два класса Error и Exception. Error – это ошибки среды самой java к ним мы имеем очень ограниченный доступ. Exception есть проверяемые ошибки, которые проверяет среда при компиляции кода и RuntimeException, которые джава не проверяет и которые могут возникнуть при работе программы