1. Назовите принципы ООП и расскажите о каждом.  
2. Дайте определение понятию “класс”.  
3. Что такое поле/атрибут класса?  
4. Как правильно организовать доступ к полям класса?  
5. Дайте определение понятию “конструктор”.

5a. В чём отличие «конструктора» от «метода»?  
6. Чем отличаются конструкторы по умолчанию, копирования и конструктор с параметрами?  
7. Какие модификации уровня доступа вы знаете, расскажите про каждый из них.  
8. Расскажите об особенностях класса с единственным закрытым (private) конструктором.  
9. О чем говорят ключевые слова “this”, “super”, где и как их можно использовать?  
10. Дайте определение понятию “метод”.  
11. Что такое сигнатура метода?  
12. Какие методы называются перегруженными?  
13. Могут ли нестатические методы перегрузить статические?  
14. Расскажите про переопределение методов.  
15. Может ли метод принимать разное количество параметров (аргументы переменной длины)?  
16. Можно ли сузить уровень доступа/тип возвращаемого значения при переопределении метода?  
17. Как получить доступ к переопределенным методам родительского класса?  
18. Какие преобразования называются нисходящими и восходящими?  
19. Чем отличается переопределение от перегрузки?  
20. Где можно инициализировать статические/нестатические поля?

**1. Назовите принципы ООП и расскажите о каждом.**

Объе́ктно-ориенти́рованноепрограмми́рование (ООП) — это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Основные принципы ООП: абстракция, инкапсуляция, наследование, полиморфизм.

**Абстракция** —  означает выделение значимой информации и исключение из рассмотрения незначимой. С точки зрения программирования это правильное разделение программы на объекты. Абстракция позволяет отобрать главные характеристики и опустить второстепенные.

Пример: описание должностей в компании. Здесь название должности значимая информация, а описание обязанностей у каждой должности это второстепенная информация. К примеру главной характеристикой для «директор» будет то, что это должность чем-то управляет, а чем именно (директор по персоналу, финансовый директор, исполнительный директор) это уже второстепенная информация.

**Инкапсуляция** — свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе. Для Java корректно будет говорить, что инкапсуляция это «сокрытие реализации». Пример из жизни — пульт от телевизора. Мы нажимаем кнопочку «увеличить громкость» и она увеличивается, но в этот момент происходят десятки процессов, которые скрыты от нас. Для Java: можно создать класс с 10 методами, например вычисляющие площадь сложной фигуры, но сделать из них 9 private. 10й метод будет называться «вычислитьПлощадь()» и объявлен public, а в нем уже будут вызываться необходимые скрытые от пользователя методы. Именно его и будет вызывать пользователь.

**Наследование** — свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс — потомком, наследником, дочерним или производным классом.

**Полиморфизм**— свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта. Пример (чуть переделанный) из ThinkinginJava:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | public interface Shape {      void draw();      void erase();  }  public class Circle implements Shape {      public void draw() {          System.out.println("Circle.draw()");      }  }  public class Triangle implements Shape {      public void draw() {          System.out.println("Triangle.draw()");      }  }    public class TestPol {        public static void main(String[] args) {          Shape shape1 = new Circle();          Shape shape2 = new Triangle();          testPoly(shape1);          testPoly(shape2);      }        public static void testPoly(Shape shape) {          shape.draw();      }  }  //Вывод в консоль:  //Circle.draw()  //Triangle.draw() |

Есть общий интерфейс «Фигура» и две его реализации «Треугольник» и «Круг». У каждого есть метод «нарисовать». Благодаря полиморфизму нам нет нужды писать отдельный метод для каждой из множества фигур, чтобы вызвать метод «нарисовать».  Вызов полиморфного метода позволяет одному типу выразить свое отличие от другого, сходного типа, хотя они и происходят от одного базового типа. Это отличие выражается различным действием методов, вызываемых через базовый класс (или интерфейс).  
Здесь приведен пример полиморфизма (также называемый динамическим связыванием, или поздним связыванием, или связыванием во время выполнения), в котором продемонстрировано как во время выполнения программы будет выполнен тот метод, который принадлежит передаваемому объекту.

Если бы не было полиморфизма и позднего связывания, то эта же программа выглядела бы примерно так:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public static void testPolyCircle(Circle circle) {          circle.draw();      }      public static void testPolyTriangle(Triangle triangle) {          triangle.draw();      } |

Т.е. для каждого класса (фигуры) мы бы писали отдельный метод. Здесь их два, а если фигур (классов) сотни?

#### 2. Дайте определение понятию “класс”.

Класс – это шаблон, описывающий общие свойства и структуру объектов данного класса. Этими свойствами могут быть как характеристики объектов (размер, вес, цвет и т.п.), так и поведения, роли и т.п.

#### 3. Что такое поле/атрибут класса?

Поле (атрибут) класса — это характеристика объекта. Например для фигуры это может быть название, площадь, периметр.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | publicclassCircleimplementsShape{        privateStringname;      privateDoublearea;      privateStringperimeter;    } |

#### 4. Как правильно организовать доступ к полям класса?

Модификатор доступа — private. Доступ через методы get\set. Сеттеры лучше не импользовать.

#### 5. Дайте определение понятию “конструктор”.

Конструктор — это специальный метод, который вызывается при создании нового объекта. Конструктор инициализирует объект непосредственно во время создания. Имя конструктора совпадает с именем класса, включая регистр, а по синтаксису конструктор похож на метод без возвращаемого значения.

|  |  |
| --- | --- |
| 1  2  3 | publicclassCircleimplementsShape{     publicCircle(){ }  } |

#### 5а. В чём отличие “конструктора” от метода.

Конструктор в отличе от методов, НЕ возвращает никакого значения(даже void). Не может быть static . А так же final, поскольку конструктор не является членом класса и не учавствует в переопределении.

#### 6. Чем отличаются конструкторы по умолчанию, копирования и конструктор с параметрами?

Конструктор по умолчанию не принимает никаких параметров. Конструктор копирования принимает в качестве параметра объект класса. Конструктор с параметрами принимает на вход параметры (обычно необходимые для инициализации полей класса).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | //конструктор по умолчанию      publicCircle(){      }        //конструктор копирования      publicCircle(Circlecircle){          this(circle.getName(),circle.getArea(),circle.getPerimeter());//будет вызван конструктор с параметрами ниже      }        //конструкторспараметрами      publicCircle(String name,Double area,String perimeter){          this.name=name;          this.area=area;          this.perimeter=perimeter;      } |

Обращаю внимание, что тема копирования (clone()) достаточно глубокая с возможностью возникновения множества неявных проблем. Немного можно почитать здесь http://habrahabr.ru/post/246993/.

#### 7. Какие модификации уровня доступа вы знаете, расскажите про каждый из них.

* private (закрытый) — доступ к члену класса не предоставляется никому, кроме методов этого класса. Другие классы того же пакета также не могут обращаться к private-членам.
* package-private, friendly, доступ по умолчанию, когда никакой модификатор не присутствует — член класса считается открытым внутри своего собственного пакета, но не доступен для кода, расположенного вне этого пакета.Т.е. если package2.Class2 extends package1.MainClass, то в Class2 методы **без идентификатора** из MainClass видны **не будут**.
* protected (защищённый) — доступ в пределах пакета и классов наследников. Доступ в классе из другого пакета будет к методам public и protected главного класса. Т.е. если package2.Class2 extends package1.MainClass, то внутри package2.Class2 методы с идентификатором protected из MainClass будут видны.
* public (открытый) — доступ для всех из любого другого кода проекта

Модификаторы в списке расположены по возрастающей видимости в программе.

#### 8. Расскажите об особенностях класса с единственным закрытым (private) конструктором.

Речь о УТИЛИТНЫХ КЛАССАХ, экземпляр которого создается 1шт на класслоадер. Как правило в такие классы помечены как final и в имени присутствует util/utils(ex: FileUtils). Невозможно создать объект класса у которого единственный private конструктор за пределами класса. Поэтому нельзя унаследоваться от такого класса. При попытке унаследоваться будет выдаваться ошибка: ThereisnodefaultconstructoravailableinимяКласса. А при попытке создать объект этого класса: ИмяКласса() hasprivateaccessinИмяКласса

#### 9. О чем говорят ключевые слова “this”, “super”, где и как их можно использовать?

super — используется для обращения к базовому классу, а this к текущему. Пример:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | publicclassAnimal{        publicvoideat(){          System.out.println("animal eat");      }  }    publicclassDogextendsAnimal{        publicvoideat(){          System.out.println("Dog eat");      }      publicvoidthisEat(){          System.out.println("Call Dog.eat()");          this.eat();      }        publicvoidsuperEat(){          System.out.println("Call Animal.eat()");          super.eat();      }    }    publicclassTest{        publicstaticvoidmain(String[]args){          Dog dog=newDog();          dog.eat();          dog.thisEat();          dog.superEat();      }  }  Dog eat  Call Dog.eat()  Dog eat  Call Animal.eat()  Animaleat |

Если написать super(), то будет вызван конструктор базового класса, а если this(), то конструктор текущего класса. Это можно использовать, например, при вызове конструктора с параметрами:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | publicDog(){          System.out.println("Call empty constructor");      }        publicDog(Stringname){          System.out.println("Call constructor with Name");          this.name=name;        }        publicDog(Stringname,Doubleweight){          this(name);          this.weight=weight;          System.out.println("Call constructor with Name and Weight");      }  }  ..  publicstaticvoidmain(String[]args){  Dog dog1=newDog("name",25.0);  }  //Вывод  Call constructor with Name  Call constructor with Name andWeight |

#### 10. Дайте определение понятию “метод”.

Метод — это последовательность команд, которые вызываются по определенному имени. Можно сказать что это функция и процедура (в случае void метода).

#### 11. Что такое сигнатура метода?

Сигнатура метода в Java — это имя метода плюс параметры (причем порядок параметров имеет значение).  
В сигнатуру метода не входит возвращаемое значение, бросаемые им исключения, а также модификаторы.

Ключевые слова public, protected, private, abstract, static, final, synchronized, native, strictfp в т.ч. аннотации для метода — это модификаторы и не являются частью сигнатуры.

!!! Часть Throws открыта к сужению(уменьшению кол-ва исключений в текущей реализации метода),но закрыта к расширению. Т.е. в классе-наследнике НЕЛЬЗЯ добавить исключение, НЕ ДОБАВИВ ЕГО В СУПЕРКЛАСС.

##### **http://docs.oracle.com/javase/specs/jls/se8/html/jls-8.html#jls-8.4.2**

#### 12. Какие методы называются перегруженными?

Java позволяет создавать несколько методов с одинаковыми именами, но разными параметрами. Создание метода с тем же именем, но с другим набором параметров называется перегрузкой. Какой из перегруженных методов должен выполняться при вызове, Java определяет на основе фактических параметров.

|  |  |
| --- | --- |
| 1  2  3  4  5 | publicvoidmethod(){}    publicvoidmethod(inta){}    publicvoidmethod(Stringstr){} |

#### 13. Могут ли нестатические методы перегрузить статические?

Не совсем. Это будут просто два разных метода для программы. Статический будет доступен по имени класса.

#### 14. Расскажите про переопределение методов. Могут ли быть переопределены статические методы?

В переопределении учавствуют ТОЛЬКО ЧЛЕНЫ КЛАССА : поля, нестатические методы…

Метод в классе-наследнике, совпадающий по сигнатуре с методом из родительского класса называется переопределенным методом. Переопределить базовый статический метод нельзя, т.к.он не явл.членом класса: InstancemethodимяМетодаinклассНаследникcannotoverridemethodимяМетодаinродительскийКласс

#### 15. Может ли метод принимать разное количество параметров (аргументы переменной длины)?

Да. Запись имеет вид method(type … val). Например publicvoidmethod(String … strings), где strings это массив, т.е. можно записать

|  |  |
| --- | --- |
| 1  2  3  4  5 | publicvoidmethod(String...strings){          for(Strings:strings){            }      } |

#### 16. Можно ли сузить уровень доступа/тип возвращаемого значения при переопределении метода?

При переопределении метода нельзя сузить модификатор доступа к методу (например с public в MainClass до private в ClassextendsMainClass). Изменить тип возвращаемого значения при переопределении метода нельзя, будет ошибка attemptingtouseincompatiblereturntype. Но можно сузить возвращаемое значение, если они совместимы. Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | publicclassAnimal{        publicAnimal eat(){          System.out.println("animal eat");          returnnull;      }        publicLongcalc(){          returnnull;      }    }  publicclassDogextendsAnimal{        publicDog eat(){          returnnewDog();      }  /\*attempting to use incompatible return type      public Integer calc() {          return null;      }  \*/  } |

#### 17. Как получить доступ к переопределенным методам родительского класса?

super.method();

Пример: Некая Map<> создаётся в методе родительского класса. В наследнике переопределённый метод может вызвать метод суперкласса для получения базовой Мар и дополнить её при необходимости своими значениями.

#### 18. Какие преобразования называются нисходящими и восходящими?

Преобразование от потомка к предку называется восходящим, от предка к потомку — нисходящим. Нисходящее преобразование должно указываться явно с помощью указания нового типа в скобках.

Например:

|  |  |
| --- | --- |
| 1  2  3  4 | Animal dog = new Dog();//восходящее преобразование. Будет потерян доступ ко всем методам, которые есть только у класса Dog.    Int x=100;  bytey=(byt e)x;  //нисходящее преобразование. Должно быть указано явно |

#### 19. Чем отличается переопределение от перегрузки?

Переопределение используется тогда, когда вы переписываете (переделываете, переопределяете) УЖЕ существующий метод. Перегрузка — это использование одного имени, но с разными входными параметрами. Например нам нужно, чтобы метод toString() для нашего класса выдавал какой-то осмысленный текст. Тогда мы переопределяем метод из класса Object и реализуем этот метод так, как нам это нужно.

|  |  |
| --- | --- |
| 1  2  3  4 | *@Override*  publicStringtoString(){  return"Хочу чтобы писался текст, а не название класса@2234SD!"  } |

Тогда как перегрузка обычно используется, чтобы не придумывать каждый раз новое имя, когда методы отличаются только входными параметрами. При перегрузке необходимый метод определяется на этапе компиляции на основе сигнатуры вызываемого метода, тогда как при переопределении нужный метод будет выявлен во время выполнения исходя из реального типа объекта.

#### 20. Где можно инициализировать статические/нестатические поля?

Статические поля можно инициализировать при объявлении, в статическом или динамическом блоке инициализации.

!!! Через СЕТТЕР статическое поле (НЕ final !!!) можно проинициализировать, НО НЕ СТОИТ. Статическое поле не относится к конкретному классу, поэтому при изменении ТАКОГО ПОЛЯ через сеттер, произойдёт измененение значения ВО ВСЕХ далее созданных инстансах, а => это уже получатся совсем другие объекты !!!

Нестатические поля можно инициализировать при объявлении, в динамическом блоке инициализации , в конструкторе и в сеттере «set()». Если поле public, то его можно инициализировать ИЗВНЕ в других классах.