IT-Academy

# Объекно-ориентированное программирование

# Рассматриваемые вопросы

IT-Academy

- Процедурное и объектно-ориентированное программирование
- Принципы ООП
- Классы и объекты
- Конструктор в Java
- Ключевое слово this в Java
- Перегрузка
- Стэк и очередь
- Передача объектов в методы
- Java varargs

# Рассматриваемые вопросы

IT-Academy

- Рекурсия
- Сборщик мусора и метод finalize
- Наследование
- Ключевое слово super в Java
- Модификаторы доступа
- Геттеры и сеттеры
- Переопределение методов
- Абстрактные классы
- Ключевое слово final

# Процедурное и объектно-ориентированное программирование

IT-Academy

Выделяют две основные методики программирования:

- процедурное
- объектно-ориентированное программирование (ООП)

# Процедурное программирование

IT-Academy

**Процедурное программирование** - это тип программирования, в котором инструкции для решения задачи выполняются одна за другой, сверху вниз, иногда возникают изменения в их последовательности. Когда программа становится более сложной на помощь приходят методы. Но современные программы настолько сложны, что даже разделение на методы не делают программу проще. И здесь на помощь приходит объектно-ориентированное программирование. Все программы, кроме кота, которые мы рассматривали до сих пор, написаны в процедурном стиле. Даже несмотря на то, что весь код прописан в классе.

# Объектно-ориентированное программирование

IT-Academy

**Объектно-ориентированное программирование (ООП)** - методика программирования, в которой основными концепциями являются понятия объектов и классов. Прежде чем начать писать инструкции для решения задачи, в задаче выделяются объекты и описываются с помощью классов. В классе прописывается поведение объектов с помощью методов и характеристики или свойства объекта с помощью переменных класса. Одной из ключевых особенностей языка Java является ООП.

# Принципы ООП

IT-Academy

***Инкапсуляция*** — это свойство системы, позволяющее объединить данные и методы в классе, и скрыть детали реализации от пользователя.

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

***Полиморфизм*** — буквально означает много форм. Это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта. “Один интерфейс, множество методов”. Реализации полиморфизма в языке Java - это перегрузка и переопределение методов, интерфейсы.

***Абстракция данных*** — это способ выделить набор значимых характеристик объекта, исключая из рассмотрения не значимые. Соответственно, абстракция — это набор всех таких характеристик.

# Что такое объект и класс?

IT-Academy

**Класс** - это шаблон для создания объекта, а **объект** - это экземпляр класса. Класс определяет структуру и поведение, которые будут совместно использоваться набором объектов. Класс содержит переменные и методы, которые называются элементами класса, членами класса. Он составляет основу инкапсуляции в Java. Каждый объект данного класса содержит структуру и поведение, которые определены классом. Иногда объекты называют экземплярами класса.

**Методы** используются для описания того, что объект класса умеет делать или что можно с ним сделать. **Переменные** - для описания свойств или характеристик объекта.

# Как создать класс

IT-Academy

**Рассмотрим как создать класс в языке Java.**
**Упрощенная общая форма определения класса:**

```
class ИмяКласса{
      тип переменнаяЭкземпляра1;
      тип переменнаяЭкземпляра2;
      // ...
      тип переменнаяЭкземпляраN;
      тип имяМетода1 ( список параметров) { // тело метода }
      тип имяМетода2 (список параметров) { // тело метода }
      тип имяМетодаN (список параметров ) { // тело метода }
}
```

```
public class Box {
    double width;
    double height;
    double depth;
}
```

# Создание объекта

IT-Academy

Объявление класса создает только шаблон, но не конкретный объект. Чтобы создать объект класса Box в Java, нужно воспользоваться оператором наподобие следующего:

```
Box myBox = new Box();
```

При создании экземпляра класса, создается объект, который содержит **собственную** копию каждой переменной экземпляра, определенной в данном классе. Создание объектов класса представляет собой двух этапный процесс:

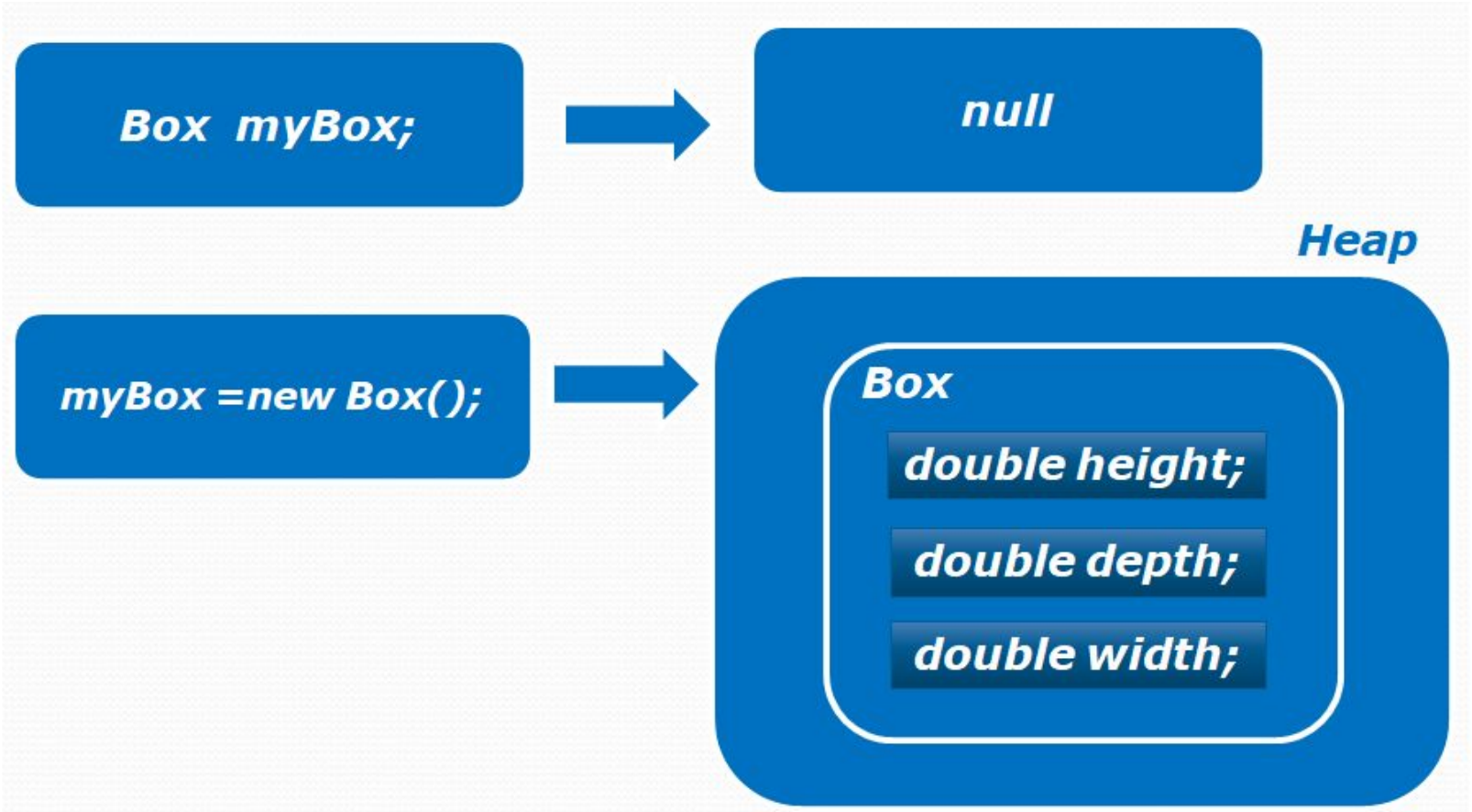
Объявление переменной типа класса. Эта переменная не определяет объект. Она является лишь переменной, которая ссылается на объект:

```
Box myBox;
```

Создание объекта. С помощью оператора new динамически (то есть во время выполнения) резервируется память для объекта и возвращается ссылка

```
myBox = new Box();
```

# Создание объекта



# Создание объекта

IT-Academy

После объявления объекта класса Box, всем переменным класса присваивается значение по умолчанию для заданного типа. Для того, чтобы обратиться к переменной класса и изменить ее или получить значение, используется имя переменной объекта:

```
public class BoxDemo1 {
        public static void main(String[] args) {
            Box myBox = new Box();
            // присвоить значение переменным экземпляра mybox myBox.width = 10;
            myBox.height = 20; myBox.depth = 15;
            // рассчитать объем параллелепипеда
            double volume = myBox.width * myBox.height * myBox.depth;
            System.out.println("Объем равен " + volume);
        }
}
```

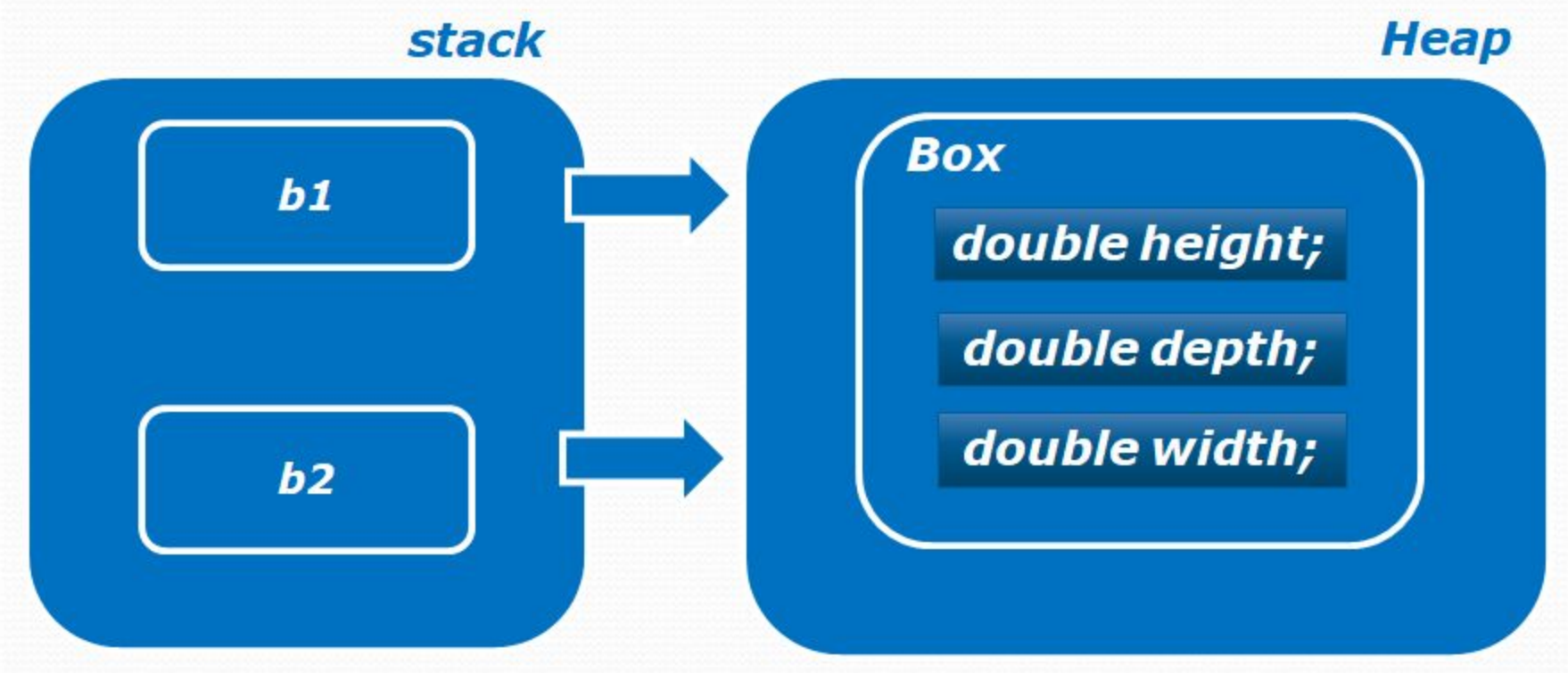
BoxDemo1

BoxDemo7

# Присваивание переменным ссылок на объекты

IT-Academy

Возможна ситуация, когда две переменные указывают на один и тот же объект в памяти:




BoxDemo6

# Добавление методов в класс

IT-Academy

Кроме переменных класс может содержать **методы**. В следующем примере в класс Box добавляется два метода:

**getVolume**() - для вычисления объема коробки и

**setDim**() - для установки размера коробки.

Обратите внимание, что теперь мы объявляем методы нестатические (без ключевого слова static). В обоих методах мы имеем доступ к переменным класса.

BoxDemo2

# Что такое конструктор?

IT-Academy

В языке Java существует такая конструкция как **конструктор**, который инициализирует объект непосредственно во время его создания. При создании объекта, то что пишется после ключевого слова new, это и есть конструктор:

```
Box myBox = new Box();
```

# Конструктор с параметрами

IT-Academy

Конструктор также как и метод может принимать на вход параметры. Такие конструкторы еще называются **параметризированными**.

Код конструктора должен заниматься только инициализацией объекта. Следует избегать вызовов из конструктора других методов, за исключением final. Метод может быть переопределен в подклассе и исказить процесс инициализации объекта.

BoxDemo4

# Ссылка на текущий объект. this

IT-Academy

Иногда требуется, чтобы метод ссылался на вызвавший его объект. Ключевое слово this в Java используется в теле любого метода для ссылки на текущий объект.

Рассмотрим конструктор, в котором параметры имеют те же имена, что и переменные класса. В этом случае параметры перекрывают область видимости переменных класса и мы не можем напрямую обратится к переменным класса. Чтобы это сделать используется ключевое слово this:

```
Box(double width, double height, double depth) {
        this.width = width;
        this.height = height;
        this.depth = depth;
}
```

# this() в конструкторе

IT-Academy

Второй вариант использования ключевого слова this() - с его помощью можно вызвать один конструктор из другого. Вызов this() может находиться только в первой строчке конструктора. Toy, ToyDemo

# Перегрузка методов

IT-Academy

В Java разрешается в одном и том же классе определять два или более метода с одним именем, если только объявления их параметров отличаются. Это называется **перегрузкой методов**.

Overloading1, Overloading2

Как же JVM различает какой метод необходимо вызвать? Для этого в Java используется тип и/или количество аргументов метода.

***Перегрузка методов*** является одним из способов поддержки **полиморфизма** в Java.

Возвращаемые типы перегружаемых методов могут отличаться, но самого возвращаемого типа недостаточно для того, чтобы отличать два разных варианта метода.

# Перегрузка конструкторов

IT-Academy

Конструкторы похожи на методы, поэтому они тоже могут быть перегружены - вы можете объявлять в одном классе несколько конструкторов, которые различаются количеством и типом переменных. В следующем примере добавлены три конструктора в класс Box6. При создании объекта вызывается только один из них - тот, который кажется наиболее подходящим.

# Память. Куча

IT-Academy

Структура памяти в Java достаточно сложна, но на начальном этапе обучения изучим две ее области: **стек** (**stack**) и **куча** (**heap**).

***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.

Любой объект, созданный в куче, имеет глобальный доступ и на него могут ссылаться из любой части приложения.

TestDemo, ReturnObject

# Память. Стэк

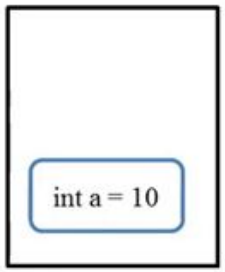
Стековая память в Java работает по схеме **LIFO** (Последний-зашел-Первый-вышел). Всякий раз, когда вызывается метод, в памяти стека создается новый блок, который содержит примитивы и ссылки на другие объекты в методе. Как только метод заканчивает работу, блок также перестает использоваться, тем самым предоставляя доступ для следующего метода.

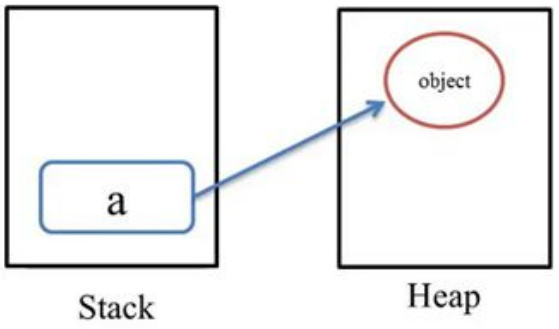
Размер стека намного меньше объема памяти в куче.

Всякий раз, когда создается объект, он всегда хранится в куче, а в памяти стека содержится ссылка на него. Память стека содержит только локальные переменные примитивных типов и ссылки на объе

```
int a = 10; // local variable
```

```
Test a = new Test( );
```



# varargs

IT-Academy

В языке Java существуют методы, которые могут принимать переменное количество аргументов. Они называются методами с аргументами переменной длины (var-args).

Для указания аргументов переменной длины служат три точки ( . . . ). Например:

```
static void test(int... array)
```

Наряду с параметром переменной длины у метода могут быть и "обычные" параметры. Но параметр переменной длины должен быть последним среди всех параметров, объявляемых в методе. Например:

```
static void test(double d, int... array)
```

VarArgs,VarArgs2,VarArgs3

# Рекурсия

IT-Academy

**Рекурсия** - это средство, которое позволяет методу вызывать самого себя. Такой метод называется рекурсивным.

Когда рекурсивный метод вызывает самого себя, новым локальным переменным и параметрам выделяется место в стеке и код метода выполняется с этими новыми исходными значениями. При каждом возврате из вызова рекурсивного метода прежние локальные переменные и параметры удаляются из стека, а выполнение продолжается с точки вызова в самом методе.

# Сборщик мусора и метод finalize

IT-Academy

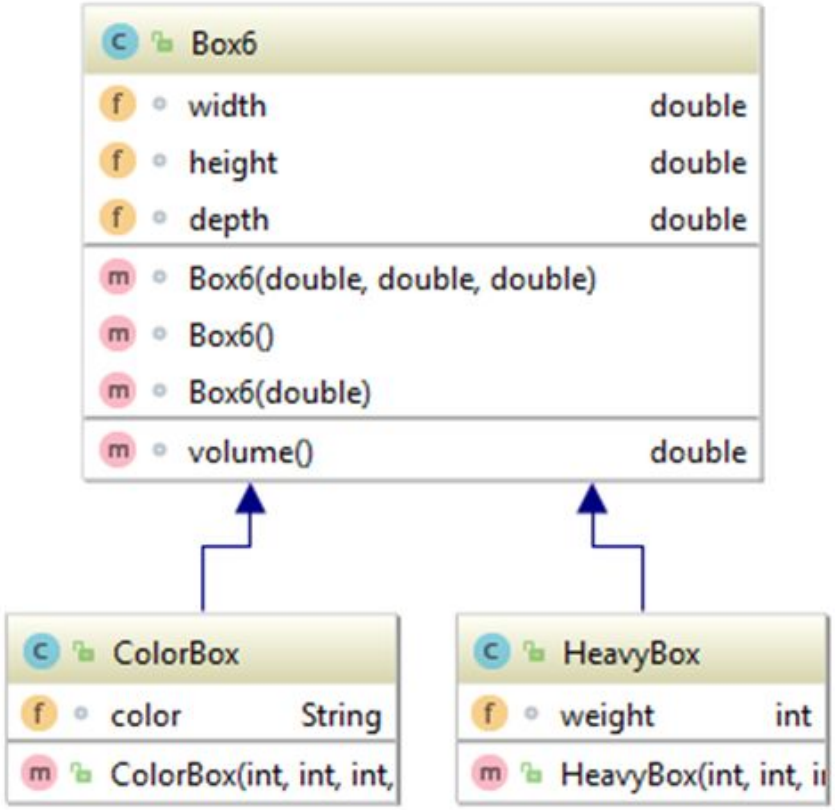
Каждый раз при создании объекта под него выделяется память. Память не резиновая и может закончится. В некоторых языках программирования разработчики должны сами контролировать освобождение памяти. В Java же освобождение памяти выполняется автоматически. Используемая для выполнения этой задачи технология называется сборщиком мусора. Есть возможность запросить сборку мусора самим, для чего используется метод **System.gc().** НО - JVM сама решит выполнять ли ваш запрос!

Сборка мусора проходит следующим образом: при отсутствии каких либо ссылок на объект программа заключает, что этот объект больше не нужен, и занимаемую объектом память можно освободить.

Метод **finalize**() не вызывается при выходе объекта из области действия. Заранее неизвестно, когда будет (и будет ли вообще) выполняться метод finalize(). И самое главное - начиная с Java 9 этот метод не рекомендуется к использованию. ( Spoon, Cup)

# Что такое наследование?

Наследование - свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью.



# Наследование

IT-Academy

Общая форма объявления класса, который наследуется от суперкласса:

```
class имяПодкласса extends имяСуперКласса {
// тело класса
}
```

DifferentBoxDemo1

Задание.

Создать Класс Product, добавить 2 поля, 2 метода.

Создать 2 класса потомка.

# Доступ к членам класса и наследование

IT-Academy

Несмотря на то, что подкласс включает в себя все члены своего суперкласса, он не может иметь доступ к тем членам суперкласса, которые объявлены как private:

```
public class A {
        public int value1;
        private int value2;
}
```

Из класса B, который является наследником класса A, невозможно напрямую обратиться к private переменной класса A. Доступ к ним можно получить через геттер методы:

```
public class B extends A {
        public int sum() {
        // return value1 + value2;
        return value1 + getValue2();
    }
}
```

# Переменная суперкласса может ссылаться на объект подкласса

IT-Academy

Ссылочной переменной суперкласса может быть присвоена ссылка на любой его подкласс.

Например, переменная heavyBox объявлена как Box, но она указывает на объект типа HeavyBox:

```
Box heavyBox = new HeavyBox(15, 10, 20, 5);
Box colorBox = new ColorBox(25, 12, 20, "красный");
```

Обратное не верно! **Нельзя** написать так:

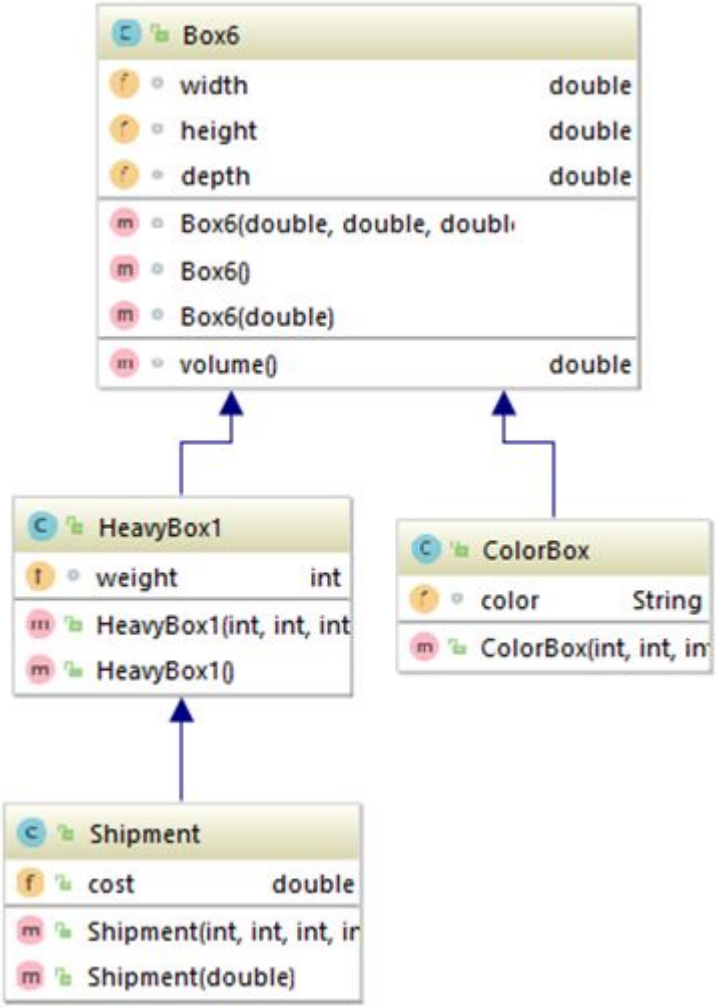
HeavyBox heavyBox = new Box(15, 10, 20).

DifferentBoxDemo2

# Создание многоуровневой иерархии

IT-Academy

Можно строить иерархии, состоящие из любого количества уровней наследования. Например, добавим класс Shipment, расширяющий HeavyBox:



# Порядок вызова конструкторов в многоуровневой иерархии

IT-Academy

В иерархии классов конструкторы выполняются в порядке наследования, начиная с суперкласса и кончая подклассом.

CallingConstructors

# Использование ключевого слова super

Ключевое слово super в Java используется когда подклассу требуется сослаться на его непосредственный суперкласс.

У ключевого слова super имеются две общие формы:

- Для вызова конструктора суперкласса: **super(списокАргументов);**
- Для обращения к члену суперкласса, скрываемому членом подкласса:

**super.member;**

# Вызов конструкторов суперкласса через super

IT-Academy

Если в иерархии классов требуется передать параметры конструктору суперкласса, то все подклассы должны передавать эти параметры вверх по иерархии. То есть из конструктора подкласса надо вызвать конструктор суперкласса с помощью **super**(). Когда метод **super**() вызывается из подкласса, вызывается конструктор его непосредственного суперкласса. Это справедливо даже для многоуровневой иерархии.

Вызов метода **super**() должен быть всегда в первом операторе, выполняемом в теле конструктора подкласса.

!Если в конструкторе наследника нет явного вызова super().

HeavyBox1

# Обращения к члену суперкласса через super

С помощью ключевого слова **super** можно обратиться к члену суперкласса из класса наследника. Чаще всего это можно сделать не используя super, но в этом примере рассмотрим случаи, когда без него не обойтись.

UseSuper

# Модификаторы досутпа

IT-Academy

Очень часто в Java доступ к некоторым членам класса желательно ограничить. Для этого и нужны модификаторы доступа, которые могут присутствовать в объявлении члена класса.

Ограничение уровня доступа к членам класса - это еще один механизм реализации принципа инкапсуляции.

# Модификаторы досутпа

IT-Academy

Существует три модификатора доступа: public, private и protected и четыре уровня доступа:

- **public** (открытый) - когда член объявляется с модификатором доступа public, он становится доступным из любого другого кода.
- **private** (закрытый) - когда член класса объявляется с модификатором доступа private, он доступен только другим членам этого же класса.
- **protected** (защищенный) - применяется только при наследовании.
- **package-friendly** - в отсутствие модификатора доступа по умолчанию член класса считается открытым в своем пакете, но недоступным для кода, находящегося за пределами этого пакета.

Modificators, ModificatorsDemo2, ModificatorsDemo3, Parent, Child

# Уровни доступа для класса

IT-Academy

Для класса, не являющегося вложенным, может быть указан только один из двух возможных уровней доступа:

- **package-friendly(default)** - если у класса имеется уровень доступа по умолчанию, такой класс оказывается доступным только для кода из данного пакета.
- **открытый (public)** - если класс объявлен как public, он доступен из любого другого кода.

Когда мы говорим, что код из одного класса (class A) имеет доступ к коду из другого класса (class B), это означает что класс A может делать одну из трех вещей:

- создать экземпляр класса B
- наследовать класс B
- иметь доступ к определенным членам класса B

# Что такое переопределение методов

IT-Academy

Если в иерархии классов совпадают имена и сигнатуры типов методов из подкласса и суперкласса, то говорят, что метод из подкласса *переопределяет* метод из суперкласса.

***Переопределение* методов** выполняется только в том случае, если имена и сигнатуры типов обоих методов одинаковы. В противном случае оба метода считаются перегружаемыми.

***Переопределение методов*** это одна из форм реализации полиморфизма, который позволяет определить в общем классе методы, которые станут общими для всех производных от него классов, а в подклассах - конкретные реализации некоторых или всех этих методов.

Override package

# Методы подставки

IT-Academy

После выхода Java 5 появилась возможность при переопределении методов указывать другой тип возвращаемого значения, в качестве которого можно использовать только типы, находящиеся ниже в иерархии наследования, чем исходный тип. Такие типы еще называются **ковариантными**.

# Переопределение и статические методы

IT-Academy

Статические методы не могут быть переопределены. Класс наследник может объявлять метод с такой же сигнатурой, что и суперкласс, но это не будет переопределением. При вызове переопределенного метода JVM выбирает нужный вариант основываясь на типе объекта. Вызов же статического метода происходит без объекта. Версия вызываемого статического метода всегда определяется на этапе компиляции.

При использовании ссылки для доступа к статическому члену компилятор при выборе метода учитывает тип ссылки, а не тип объекта, ей присвоенного.

Package statisc

# Переопределение методов в классах наследниках

IT-Academy

Методы объявленные как private никто, кроме самого класса не видит. Поэтому их наличие/отсутствие никак не отражается на классах-наследниках. Они с легкостью могут объявлять методы с такой же сигнатурой и любыми модификаторами. Но это плохой тон! Также класс наследник может расширить видимость protected метода до public. Сузить видимость класс-наследник не может.

# Аннотация @Override

IT-Academy

Необязательная аннотация @Override используется с методом для указания того, что он переопределен. Если метод переопределен неверно, код не скомпилируется.

# Вопрос

IT-Academy

IT-Academy

# Спасибо за внимание