Объектно-ориентированное программирование в JAVA

Наследование

Принцип наследования состоит в том, что новые классы можно создавать из уже существующих классов.

При наследовании методы и поля существующего класса используются повторно (наследуются) вновь создаваемым классом, причем для адаптации нового класса к новым условиям в него добавляют дополнительные поля и методы.

Ссылка на материал для дополнительного изучения

Хорстман К. «JAVA Библиотека профессионалов Том 1. Основы Одиннадцатое издание» Глава 5

Классы, суперклассы и подклассы

Ключевое слово **extends** означает, что на основе существующего класса создается новый класс.

Существующий класс называется ***суперклассом***, *базовым* или *родительским*, а вновь создаваемый — ***подклассом***, *производным* или *порожденным*.

В среде программирующих на Java наиболее широко распространены термины ***суперкласс*** и ***подкласс***, хотя некоторые из них предпочитают пользоваться терминами ***родительский*** и ***порожденный***, более тесно связанными с понятием наследования.

Ссылка на материал для дополнительного изучения

Хорстман К. «JAVA Библиотека профессионалов Том 1. Основы Одиннадцатое издание» Глава 5.1.1

Переопределение методов

```
public double getSalary()


{
      double baseSalary = super.getSalary();

   return baseSalary + bonus;
}
```

Вызов метода суперкласса должен осуществляться с помощью ключевого слова super

Пояснение. Если в переопределяемом методе используется метод суперкласса, то он должен вызываться вместе с ключевым словом **super**.

В подкласс можно *вводить* поля, а также *вводить* и *переопределять* методы из суперкласса.

Ссылка на материал для дополнительного изучения

Хорстман К. «JAVA Библиотека профессионалов Том 1. Основы Одиннадцатое издание» Глава 5.1.2

Конструкторы подклассов

```
public Manager(String n, double s)
{
   super(n, s);

   bonus = 0;
}
```

Вызов конструктора суперкласса **Employee** с параметрами **n**, **s**.

Конструктор класса **Manager** не имеет доступа к закрытым полям класса **Employee,** поэтому он должен инициализировать их, вызывая другой конструктор с помощью ключевого слова **super.**

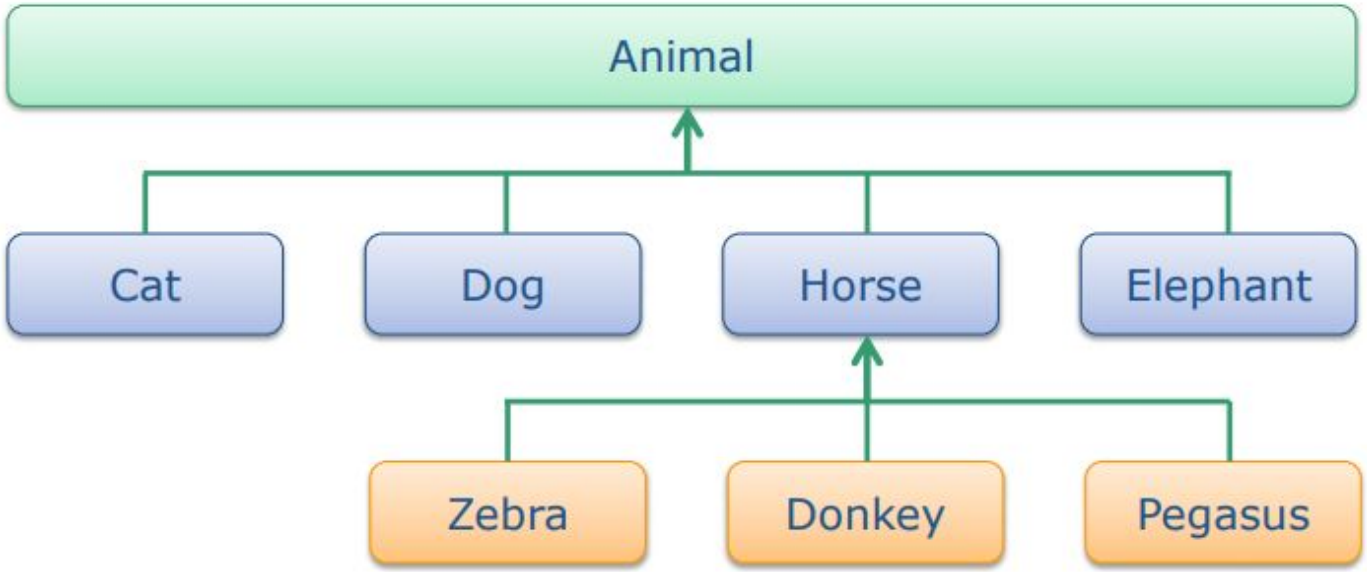
Вызов, содержащий обращение **super,** должен быть первым оператором в конструкторе подкласса.

Ссылка на материал для дополнительного изучения

Хорстман К. «JAVA Библиотека профессионалов Том 1. Основы Одиннадцатое издание» Глава 5.1.3

Иерархия наследования

- ✓ Наследование не обязательно ограничивается одним уровнем классов.
- ✓ Совокупность всех классов, производных от общего суперкласса, называется иерархией наследования.
- ✓ Путь от конкретного класса к его потомкам в иерархии называется **цепочкой наследования**.



Ссылка на материал для дополнительного изучения

Хорстман К. «JAVA Библиотека профессионалов Том 1. Основы Одиннадцатое издание» Глава 5.1.4

Полиморфизм

Существует простое правило, позволяющее определить, стоит ли в конкретной ситуации применять наследование или нет.
Если между объектами существует отношение "**является**", то каждый объект подкласса *является* объектом суперкласса.
Например, каждый руководитель является работником.

```
Manager boss = new manager(...);
Employee[] staff = new Employee[3];
staff[0] = boss;

boss.setBonus(5000); // Допустимо!
staff[0].setBonus(5000); // ОШИБКА!
// не все работники являются руководителями
```

Ссылка на материал для дополнительного изучения

Хорстман К. «JAVA Библиотека профессионалов Том 1. Основы Одиннадцатое издание» Глава 5.1.5

Предотвращение наследования

Иногда наследование оказывается нежелательным. Классы, которые нельзя расширить, называются *конечными.*

Для указания на это в определении класса используется модификатор доступа **final.**

Допустим, требуется предотвратить создание подклассов, производных от класса **Executive.**

В таком случае класс **Executive** определяется следующим образом:

```
final class Executive extends Manager{
    ...
}
```

Ссылка на материал для дополнительного изучения

Хорстман К. «JAVA Библиотека профессионалов Том 1. Основы Одиннадцатое издание» Глава 5.1.7

Предотвращение наследования

Отдельный метод класса также может быть конечным.

Такой метод не может быть переопределен в подклассах.

```
class Employee
{
   ...
   public final String getName()
   {
      return name;
   }
      ...
}
```

С помощью модификатора доступа **final** могут быть также описаны поля, являющиеся константами. После создания объекта значение такого поля нельзя изменить.

Но если **класс** объявлен как **final**, то конечными автоматически становятся только его **методы**, но не поля.

Ссылка на материал для дополнительного изучения

Хорстман К. «JAVA Библиотека профессионалов Том 1. Основы Одиннадцатое издание» Глава 5.1.7

Приведение типов

Случается, что ссылку на объект требуется привести к типу другого класса.

Для такого приведения типов служат те же самые синтаксические конструкции, что и для числовых выражений.

```
Manager boss = (Manager) staff[0];
```

перед приведением типов следует непременно проверить его корректность:

```
if (staff[1] instanceof Manager)
        {
        boss = (Manager) staff[1];
        ...
        }
```

Оператор **instanceof** нужен, чтобы проверить, был ли объект, на который ссылается переменная X, создан на основе какого-либо класса Y.

Ссылка на материал для дополнительного изучения

Хорстман К. «JAVA Библиотека профессионалов Том 1. Основы Одиннадцатое издание» Глава 5.1.8

Абстрактные классы

Абстрактный класс необходим, когда некоторое поведение характерно для большинства или всех объектов данного класса, но некоторые аспекты имеют смысл лишь для ограниченного круга объектов, не составляющих суперкласса.

В Java такие классы объявляются с ключевым словом abstract, и каждый метод, не реализованный в классе, также объявляется abstract.

Ссылка на материал для дополнительного изучения

Хорстман К. «JAVA Библиотека профессионалов Том 1. Основы Одиннадцатое издание» Глава 5.1.9

Защищенный доступ

1. Модификатор доступа **private** — ограничивает область действия классом.
2. Модификатор доступа public — не ограничивает область действия.
3. Модификатор доступа **protected** — ограничивает область действия пакетом и всеми подклассами.
4. Модификатор доступа *отсутствует* — область действия ограничивается пакетом по умолчанию.

Ссылка на материал для дополнительного изучения

Хорстман К. «JAVA Библиотека профессионалов Том 1. Основы Одиннадцатое издание» Глава 5.1.10

Глобальный суперкласс Object

- В языке Java ЛЮБОЙ класс является неявным наследником класса Object (иначе говоря, экземпляр любого класса ЯВЛЯЕТСЯ объектом)

- Свойства класса:

    - Возможность создавать массивы (контейнеры) из объектов произвольного типа (на самом деле, контейнеры как раз хранят внутри себя ссылки типа Object)
    - Возможность сравнить два объекта любого типа на равенство
    - Возможность получить строковое представление любого объекта
    - Класс Object содержит общие свойства всех объектов Java

Ссылка на материал для дополнительного изучения

Хорстман К. «JAVA Библиотека профессионалов Том 1. Основы Одиннадцатое издание» Глава 5.2

Методы класса Object

- equals – сравнение двух ЛЮБЫХ ОБЪЕКТОВ на равенство СОДЕРЖИМОГО; по умолчанию – каждый объект равен ТОЛЬКО самому себе
- свойства операции сравнения на равенство:
  - рефлексивность – любой объект ВСЕГДА равен самому себе
  - симметричность – если x.equals(y), то y.equals(x) и наоборот
  - транзитивность – если x.equals(y) и y.equals(z), то x.equals(z)
  - никакой объект не равен null
  - консистентость
- сравнение на равенство используется в некоторых методах коллекций
- toString – формирование строкового представления объекта; по умолчанию формируется из адреса объекта
- getClass – возвращает объект типа Class, имеющий доступ к спискам полей и методов данного типа (Reflection, рефлексия, интроспекция – отслеживание собственной структуры)
- clone() – возвращает копию данного объекта
- finalize() – вызывается сборщиком мусора перед разрушением объекта

Ссылка на материал для дополнительного изучения

Хорстман К. «JAVA Библиотека профессионалов Том 1. Основы Одиннадцатое издание» Глава 5.2

Объектные оболочки и автоупаковка

Классы-оболочки Java являются Объектным представлением восьми примитивных типов в Java.

Примитивный тип	Класс-обертка	Аргументы
byte	Byte	byte или String
short	Short	short или String
int	Integer	int или String
long	Long	long или String
float	Float	float, double или String
double	Double	double или String
char	Character	char
boolean	Boolean	boolean или String

Ссылка на материал для дополнительного изучения

Хорстман К. «JAVA Библиотека профессионалов Том 1. Основы Одиннадцатое издание» Глава 5.4

Объектные оболочки и автоупаковка

У примитивных типов есть **объекты-аналоги - "классы оболочки"**, или **"wrapper"**.

Класс называется "оболочкой" потому, что он, по сути, копирует то, что уже существует, **но добавляет новые возможности для работы с привычными типами**.

Но поскольку это именно класс, он может создавать свои экземпляры.
Они будут хранить внутри нужные значения примитивов, при этом будут являться настоящими объектами.

Ссылка на материал для дополнительного изучения

Хорстман К. «JAVA Библиотека профессионалов Том 1. Основы Одиннадцатое издание» Глава 5.4

Объектные оболочки и автоупаковка

Объекты классов оберток создаются так же, как и любые другие:

```
Integer i = new Integer(682);

Double d = new Double(2.33);

Boolean b = new Boolean(false);
```

У классов-оболочек есть методы. У примитивных типов методов нет.

```
Integer i = new Integer(432);
String s = i.toString();
```

Ссылка на материал для дополнительного изучения

Хорстман К. «JAVA Библиотека профессионалов Том 1. Основы Одиннадцатое издание» Глава 5.4

Объектные оболочки и автоупаковка

Автоупаковка (autoboxing) — это автоматическая конвертация из примитивных типов в соответствующий этому типу класс-обёртку, вставляемая компилятором Java, например из **float** во **Float** , из **int** в **Integer**.

```
Character ch = 'a';
Integer i1 = 220;
Double d1 = 300.0;
Boolean b1 = false;
```

Параметры передаются в методы по-разному:
- примитивы — по значению,
- а объекты — по ссылке.

Ссылка на материал для дополнительного изучения

Хорстман К. «JAVA Библиотека профессионалов Том 1. Основы Одиннадцатое издание» Глава 5.4

Методы с переменным числом параметров

- **Определение метода**

```
public double sum(double... values) {
    double sum = 0.0;
    for (double v : values) {
        sum += v;
    }
    return sum;
}
```

- **Вызов метода**

```
sum();
sum(1.0);
sum(1.0, 2.0);
sum(1.0, 2.0, 3.0);
```

Ссылка на материал для дополнительного изучения

Хорстман К. «JAVA Библиотека профессионалов Том 1. Основы Одиннадцатое издание» Глава 5.5