Лекция 11

Приведение типов, instanceof.

Оператор instanceof

- **instanceof** – это очень простой и эффективный в использовании оператор. Он используется в виде: «объект» instanceof «класс». Он проверяет, является ли объект объектом определенного класса.

	Код на Java	Описание
1	Object o = **new** Integer(3); **boolean** isInt = o **instanceof** Integer;	**isInt** будет равно **true**. Объект, на который ссылается переменная o, является объектом класса **Integer**.
2	Object o = "Mama"; **boolean** isInt = o **instanceof** Integer;	**isInt** будет равно **false**. Объект, на который ссылается переменная o, не является объектом класса **Integer**, он является объектом класса **String**.
3	InputStream is = **new** FileInputStream(""); **boolean** isFIS = is **instanceof** FileInputStream;	**isFIS** будет равно **true**. Объект, на который ссылается переменная is, является объектом класса **FileInputStream**.

- Этот оператор учитывает и наследование.

	Код на Java	Описание

1	**class** Animal
	{
	}
	class Cat **extends** Animal
	{
	}
	class Tiger **extends** Cat
	{
	}
		Тут мы видим три объявленных класса: животное, кот и тигр. Кот наследуется от Животного. А Тигр от Кота.

2	Object o = **new** Tiger(); **boolean** isCat = o **instanceof** Cat; **boolean** isTiger = o **instanceof** Tiger; **boolean** isAnimal = o **instanceof** Animal;	**isCat** будет равно **true**. **isTiger** будет равно **true**. **isAnimal** будет равно **true**.
3	Object o = **new** Animal(); **boolean** isCat = o **instanceof** Cat; **boolean** isTiger = o **instanceof** Tiger; **boolean** isAnimal = o **instanceof** Animal;	**isCat** будет равно **false**. **isTiger** будет равно **false**. **isAnimal** будет равно **true**.

- Этот оператор учитывает и интерфейсы.

	Код на Java	Описание

1	**interface** Moveable
	{
	}
	class Cat
	{
	}
	class TomCat **extends** Cat **implements** Moveable
	{
	}
		Создадим два класса: Cat, TomCat и интерфейс Moveable

2	Cat o = **new** TomCat(); **boolean** isCat = o **instanceof** Cat; **boolean** isMoveable = o **instanceof** Moveable; **boolean** isTom = o **instanceof** TomCat;	**isCat** будет равно **true**. **isMoveable** будет равно **true**. **isTom** будет равно **true**.
3	Cat o = **new** Cat(); **boolean** isCat = o **instanceof** Cat; **boolean** isMoveable = o **instanceof** Moveable; **boolean** isTom = o **instanceof** TomCat;	**isCat** будет равно **true**. **isMoveable** будет равно **false**. **isTom** будет равно **false**.

Оператор instanceof

- Оператор instanceof имеет вид: **a instanceof B**.
- Другими словами, оператор **instanceof** вернет значение **true**, если:
- **1)** переменная **a** хранит ссылку на объект типа **B**
- **2)** переменная **a** хранит ссылку на объект, класс которого унаследован от **B**
- **3)** переменная **a** хранит ссылку на объект реализующий интерфейс **B**
- Иначе оператор **instanceof** вернет значение **false**.

Приведение типов. Расширение и сужение

- Представьте себе цепочку наследования класса: класс, его родитель, родитель родителя и т.д. до самого класса Object. Т.к. **класс содержит все методы класса, от которого он был унаследован**, то объект этого класса можно сохранить в переменную **любого из его типов родителей.**

	Код на Java	Описание

1	**class** Animal
	{
	  public void doAnimalActions(){}
	}
	class Cat **extends** Animal
	{
	  public void doCatActions(){}
	}
	class Tiger **extends** Cat
	{
	  public void doTigerActions(){}
	}
		Тут мы видим три объявленных класса: животное, кот и тигр. Кот наследуется от Животного. А Тигр от Кота.

2	**public static void** main(String[] args)
	{
	  Tiger tiger = **new** Tiger();
	  Cat cat = **new** Tiger();
	  Animal animal = **new** Tiger();
	  Object obj = **new** Tiger();
	}
		Объект класса Tiger всегда можно спокойно присвоить переменной с типом класса-родителя. Для класса Tiger – это Cat, Animal и Object.

- Теперь рассмотрим, что же такое расширение и сужение типов.

- Если в результате присваивания мы двигаемся по цепочке наследования вверх (к типу Object), то это — расширение типа (оно же — восходящее преобразование или upcasting), а если вниз, к типу объекта, то это — сужение типа (оно же — нисходящее преобразование или downcasting).

- **Движение вверх по цепочке наследования называется расширением, поскольку оно приводит к более общему типу. Но при этом теряется возможность вызвать методы, которые были добавлены в класс при наследовании.**

Код на Java	Описание

public static void main(String[] args)
{
Object obj = **new** Tiger();
Animal animal = (Animal) obj;
Cat cat = (Cat) obj;
Tiger tiger = (Tiger) animal;
Tiger tiger2 = (Tiger) cat;
}
	При сужении типа, нужно использовать оператор преобразования типа, то есть мы выполняем явное преобразование.
	
	При этом Java-машина выполняет проверку, а действительно ли данный объект унаследован от Типа, к которому мы хотим его преобразовать. Такое небольшое нововведение уменьшило количество ошибок в преобразовании типов в разы, и существенно повысило стабильность работы Java-программ.

Код на Java	Описание

public static void main(String[] args)
{
  Object obj = **new** Tiger();
  if (obj **instanceof** Cat)
  {
  Cat cat = (Cat) obj;
  cat.doCatActions();
  }
}
	Еще лучше – использовать проверку **instanceof**

Код на Java

```
public static void main(String[] args)
{
 Animal animal = new Tiger();
 doAllAction(animal);

 Animal animal2 = new Cat();
 doAllAction(animal2);

 Animal animal3 = new Animal();
 doAllAction(animal3);
}

public static void doAllAction(Animal animal)
{
 if (animal instanceof Tiger)
 {
  Tiger tiger = (Tiger) animal;
  tiger.doTigerActions();
 }

 if (animal instanceof Cat)
 {
  Cat cat = (Cat) animal;
  cat.doCatActions();
 }

 animal.doAnimalActions();
}
```

Описание

Мы (наш код) не всегда знаем, с объектом какого типа мы работаем. Это может быть как объект того же типа, что и переменная (Animal), так и любой тип-наследник (Cat, Tiger).
Рассмотрим метод doAllAction. Он корректно работает в независимости от того, объект какого типа в него передали.
Т.е. он корректно работает для всех трех типов Animal, Cat, Tiger.

Код на Java	Описание

public static void main(String[] args)
{
  Cat cat = **new** Tiger();
  Animal animal = cat;
  Object obj = cat;
}
	Тут мы видим три присваивания. Все они являются примерами расширения типа.
	
	Оператор преобразования типа тут не нужен, так как не нужна проверка. **Ссылку на объект всегда можно сохранить в переменную любого его базового типа.**

Задачи

1. Исправьте строчку 'Cat cat = new Cat();' так, чтобы программа вывела "**Bingo!**".

```
public class Solution {
   public static void main(String[] args) {
      Cat cat = new Cat();

      boolean isCat = cat instanceof Cat;
      boolean isMovable = cat instanceof CanMove;
      boolean isTom = cat instanceof TomCat;

      if (isCat && isMovable && isTom) System.out.println("Bingo!");
   }

   interface CanMove {
   }

   static class Cat implements CanMove {
   }

   static class TomCat extends Cat {

   }
}
```

2. Исправьте строчку 'Object animal = new Pet();' в методе main() так, чтобы программа вывела "**Bingo!**".

```
public class Solution {
   public static void main(String[] args) {
      Object animal = new Pet();
      boolean isCat = animal instanceof Cat;
      boolean isTiger = animal instanceof Tiger;
      boolean isPet = animal instanceof Pet;

      printResults(isCat, isTiger, isPet);
   }

   private static void printResults(boolean cat, boolean tiger, boolean pet) {
      if (cat && tiger && pet) System.out.println("Bingo!");
   }

   static class Pet {
   }

   static class Cat extends Pet {
   }

   static class Tiger extends Cat {
   }

}
```

3. Посмотрите, что делает эта программа. Затем измените haveFun так, чтобы он вызывал метод:
play(), если person имеет тип Player.
dance(), если person имеет тип Dancer.

```
public class Solution {
   public static void main(String[] args) throws Exception {
      BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

      Person person = null;
      String key;
      while (!(key = reader.readLine()).equals("exit")) {
         if ("player".equals(key)) {
            person = new Player();
         } else if ("dancer".equals(key)) {
            person = new Dancer();
         }
         haveFun(person);
      }
   }

   public static void haveFun(Person person) {
      //напишите тут ваш код
   }

   interface Person {
   }

   static class Player implements Person {
      void play() {
         System.out.println("playing");
      }
   }

   static class Dancer implements Person {
      void dance() {
         System.out.println("dancing");
      }
   }
}
```

4. В этой задаче нужно:
   1. Правильно расставить наследование между **Building** (**здание**) и **School** (**здание школы**).
   2. Подумать, объект какого класса должны возвращать методы getSchool и getBuilding.
   3. Изменить **null** на объект класса **Building** или **School**.

   Сигнатуры методов getSchool() и getBuilding() не менять.

```
public class Solution {
   public static void main(String[] args) {
      Building school = getSchool();
      Building shop = getBuilding();

      System.out.println(school);
      System.out.println(shop);
   }

   public static Building getSchool() {
      //измените null на объект класса Building или School
      return null;
   }

   public static Building getBuilding() {
      //измените null на объект класса Building или School
      return null;
   }

   static class School /*Добавьте сюда ваш код*/ {
      @Override
      public String toString() {
         return "School";
      }
   }

   static class Building /*Добавьте сюда ваш код*/ {
      @Override
      public String toString() {
         return "Building";
      }
   }
}
```