**Приведение типов. Расширение и сужение**

[Java Core](https://javarush.ru/quests/QUEST_JAVA_CORE)

[Уровень 4](https://javarush.ru/quests/lectures?quest=QUEST_JAVA_CORE&level=4), Лекция 3

— Привет, Амиго! Тема сегодняшней лекции – расширение и сужение типов. С расширением и сужением примитивных типов ты познакомился уже давно. На 10 уровне. Сегодня мы расскажем, как это работает для ссылочных типов, т.е. для объектов классов.

Тут все довольно просто на самом деле. Представь 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.**

Пример:

|  |  |
| --- | --- |
| **Код** | **Описание** |
| 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. |
| 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** | **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. |

4

Задача

Java Core,  4 уровень,  3 лекция

Набираем код Ӏ Java Core: 4 уровень, 3 лекция

Внимание! Объявляется набор кода на JavaRush. Чтобы поучаствовать, включите режим повышенной внимательности, немного разомните пальцы, затем — расслабьте их и… набирайте код в соответствующем окошке. На самом деле это довольно полезное занятие, а не пустая трата времени, как может показаться.

|  |  |
| --- | --- |
| **Код** | **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** |
| 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();  } | И вот почему. Смотрим на пример слева.  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. |
| 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.** |

— О, на предпоследнем примере все стало понятно. И для чего нужна проверка, и для чего нужно преобразование типов.

— Надеюсь, что так. Хочу обратить твое внимание на следующую вещь:

**С объектом при таком присваивании ничего не происходит! Меняется только количество методов**, которое можно вызвать с помощью **конкретной переменной-ссылки.**

Например, переменная класса Cat позволяет вызывать методы doAnimalActions & doCatActions, и ничего не знает о методе doTigerActions, даже если ссылается на объект класса Tiger.

— Ну, это-то уже ясно. Оказалось легче, чем я думал.