Лекция 7.1: ООП / Понятие о типе (ClassCastException, instanceOf, …).

1. Приведение типов и сохранение типа
….1.1 Разница между приведением примитивных типов и ссылочных типов
….1.2 Явное и неявное приведение ссылочных типов
….1.3 Номинальные и структурные системы типов

Необходимо отметить серьезную разницу между
ОДИНАКОВО
- и у примитивов и у ссылочных есть понятие преобразования вверх и преобразования вниз
- и у примитивов и у ссылочных преобразования вверх происходит неявно, а вниз – явно
РАЗЛИЧАЕТСЯ
- примитивы: возможны преобразования между любыми примитивами
- ссылочные: возможны преобразования типа исключительно в рамках его верхнего дерева

- примитивы: при любом преобразовании примитивов происходит изменение памяти, в которой расположено значение
- ссылочные: при любом преобразовании не трогают объект в хипе

- примитивы: при преобразовании “вверх” возможна потеря значения
- примитивы: при преобразовании “вниз” возможна потеря значения
- ссылочных: у значение никогда не теряется, но при преобразовании может теряется интерфейс

Test: OOP.GetClass.Basic

Для прохождения теста по теме OOP.GetClass (уровень сложности теста: Basic) нажмите “Start Quiz”

Test: OOP.GetClass.Mid

Для прохождения теста по теме OOP.GetClass (уровень сложности теста: Mid) нажмите “Start Quiz”

Test: OOP.InstanceOf.Basic

Для прохождения теста по теме OOP.InstanceOf (уровень сложности теста: Basic) нажмите “Start Quiz”

Test: OOP.InstanceOf.Mid

Для прохождения теста по теме OOP.InstanceOf (уровень сложности теста: Mid) нажмите “Start Quiz”

Test: OOP.Type.Basic

Для прохождения теста по теме OOP.Type (уровень сложности теста: Basic) нажмите “Start Quiz”

Test: OOP.Subtype.A.Basic

Для прохождения теста по теме OOP.Subtype.A (уровень сложности теста: Basic) нажмите “Start Quiz”

Test: OOP.Subtype.B.Basic

Для прохождения теста по теме OOP.Subtype.B (уровень сложности теста: Basic) нажмите “Start Quiz”

Java – сильно типизированный язык. Это значит, что к каждой выделенной под объект области памяти крепко привязан тип и он неизменен.

Основы наследования (class, interface, extends, implements)
В Java отношение предок-потомок выражается при помощи ключевых слов extends или implements у потомка. По типу желаемому типу предка и потомка однозначно выясняется – могут ли они быть предком и потомком и если могут, то каким ключевым словом это отношение выражается.
extends – расширяет (класс расширяет класс или интерфейс расширяет интерфейс)
implements – реализует (класс реализует интерфейс)
1. Если предок и потомок сущности одного типа (классы или интерфейсы) то отношение наследование – extends.
2. Если предок – интерфейс, а потомок – класс, то implements
3. Класс не может быть потомком интерфейса

Основы наследования (файлы, области видимости)
Правила расположения по файлам:
1. В файле с именем XYZ.java может быть объявлено любое количество классов/интерфейсов верхнего уровня (top level), но не более одной с модификатором public.
2. В файле с именем XYZ.java не обязан быть класс/интерфейс верхнего уровня (top level) с модификатором public.
Класс/интерфейс верхнего уровня (top level) – это класс/интерфейс объявленный вне других классов/интерфейсов.

В этом примере
TopLevelClass – класс верхнего уровня (top level)
NestedClassA – вложенный (nested) класс (объявлен внутри класса TopLevelClass)
NestedInterfaceA – вложенный (nested) интерфейс (объявлен внутри класса TopLevelClass)
NestedClassB – вложенный (nested) класс (объявлен внутри интерфейса NestedInterfaceA)

```
class TopLevelClass {
    class NestedClassA {}
    interface NestedInterfaceA {
        class NestedClassB {}
    }
}
```

Правила областей видимости:
1. Классы/интерфейсы верхнего уровня могут иметь только два уровня видимости
- public
- по умолчанию (default, package private, ???)
2. Вложенные классы/интерфейсы верхнего уровня могут все 4 уровня видимости
- public
- protected
- по умолчанию (default, package private, ???)
- private
3. При наследовании области видимости предка и потомка никак не ограничены (за исключением того, что потомок должен видеть предка) – ???.

Что проверяется в момент компиляции (compile time)
1. Автоматическое приведение потомка к предку (класс/абс.класс/интерфейс)
В тестах это обычно выглядит вот так

```
public class App {
    public static void main(String[] args) {
        Child child = new Child();
        Parent parent = child;
    }
}
```

или вот так

```
public class App {
    public static void main(String[] args) {
        Parent ref = new Child();
        // ref - ссылка типа Parent на объект типа Child
    }
}
```

А в жизни обычно при передаче в метод принимающий предка

```
public class App {
    public static void main(String[] args) {
        method(new Child());
    }
    public static void method(Parent ref) {
        // ref - ссылка типа Parent на объект типа Child
    }
}
```

Или при помещении в контейнер (массив, коллекцию) типа предка

```
public class App {
    public static void main(String[] args) {
        Parent[] array = {new Parent(), new Child()};
        for (Parent parent : array) {
            // parent - для 0-элемента ссылается на Parent, а для 1-элемента - на Ch
        }
    }
}
```

```
import java.util.ArrayList;

public class App {
    public static void main(String[] args) {
        ArrayList<Parent> list = new ArrayList<>();
        list.add(new Child());
        list.add(new Parent());
        Child child = new Child();
        list.add(child);
        for (Parent parent : list) {
            // parent для 0-элемента ссылается на Parent
            // для 1-элемента - на Child
            // для 2-элемента - на Child
        }
    }
}
```

2. Нет неявного приведение предка к потомку (класс/абс.класс/интерфейс)

```
public class App {
    public static void main(String[] args) {
        Child ref = new Parent();
    }
}

>> COMPILATION ERROR
```

Для null – тоже не работает

```
public class App {
    public static void main(String[] args) {
        Parent ref0 = null;
        Child ref1 = ref0;
    }
}

>> COMPILATION ERROR
```

3. Есть явное приведение предка к потомку (класс/абс.класс/интерфейс)

```
public class App {
    public static void main(String[] args) {
        Child ref = (Child) new Parent();
    }
}

>> ??? ClassCastException
[java]
Хотя в момент исполнения и будет ClassCastException, но тут компилятор не отвергнет
Странно, что компилятор не отвергает явное нарушение, но это ??? с более типичной с
[java]
public class App {
    public static void main(String[] args) {
        method(new Parent());
        method(new Child());
    }
    public static void method(Parent ref) {
        if (ref instanceof Child) {
            Child child = (Child) ref;
            System.out.println("It's Child");
        } else {
            System.out.println("It's NOT Child");
        }
    }
}

>> It's NOT Child
>> It's Child
```

4. Компилятор отвергает неявное приведение брата к брату (класс/абс.класс)

```
public class App {
    public static void main(String[] args) {
        ChildA refA = new ChildA();
        ChildB refB = refA;
    }
}

>> COMPILATION ERROR
```

Обман

```
xxx
```

5. Компилятор принимает явное приведение брата к брату (интерфейс)

```
xxx
```

Что проверяется в момент выполнения (runtime)

```
xxx
```

```
xxx
```

```
xxx
```

```
xxx
```

Тесты по всей лекции

Тест, состоящий из случайных вопросов тестов этой лекции

Some issues occured with quiz randomizer :(

Видео

Набор декабрь 2013 (начиная с 1 час 10 минут)
Набор октябрь 2013
Набор июль 2013
Набор апрель 2013
Набор февраль 2013
Набор январь 2013
Набор октябрь 2012

Лабораторные

???

Литература

???

Приветствуем, pafnat

Настройки профиля
Выйти

Ваша успеваемость

Java Core	14.34%
1. Основы Java	88.07%
2. Базовые алгоритмы	24.13%
3. Исключения	31.21%
4. Ввод/вывод	0%
5. Многопоточность	0%
6. Коллекции	0%
7. ООП: Синтаксис	0%
8. ООП: Шаблоны	0%
9. Продвинутые возможности	0%
10. Java 8	0%