OTUS

ОНЛАЙН-ОБРАЗОВАНИЕ

30– Working with Inheritance (Часть 3)

Дмитрий Коган

Как меня слышно и видно?

Если нет – напишите, если слышите – смайлик в чат.

Цели :

- Добьём интерфейсы
- Научимся (волшебно) кастовать
- Придём в восхищение от загораживания

Начинаем?

Темы экзамена

- ❑ Java Basics
- ❑ Working with Java Data Types
- ❑ Using Operators and Decision Constructs
- ❑ Creating and Using Arrays
- ❑ Using Loop Constructs
- ❑ Working with Methods and Encapsulation
- ❑ Working with Inheritance
- ❑ Handling Exceptions
- ❑ Working with Selected classes from the Java API

Подтемы экзамена

Working with Inheritance

- **Describe inheritance and its benefits**
- **Develop code that makes use of polymorphism; develop code that overrides methods; differentiate between the type of a reference and the type of an object**
- **Determine when casting is necessary**
- **Use super and this to access objects and constructors**
- **Use abstract classes and interfaces**

Интерфейсы и наследование

Наследование

- ✓ От своего непосредственного суперкласса класс наследует все конкретные методы (как static, так и instance)
- ✓ От своего непосредственного суперкласса и непосредственных суперинтерфейсов класс наследует все abstract- и default-методы
- ✓ Класс не наследует static-методы, объявленные в суперинтерфейсах
- ✓ Интерфейсный static-метод можно вызывать только на имени его интерфейса.

Что в имени тебе моём

```
interface Inter { static void run(){ System.out.println("Inter"); } }

class Test implements Inter{
    public void run(){ System.out.println("Test"); }
    public static void main(String[] args) {
        Inter t = new Test();
    //    t.run();                  // INVALID
        Inter.run();                // Inter
        ((Test)t).run();            // Test
    }
}
```

Упражнение

```
// File: RaceA.java
interface IJogger {
  default boolean justDoIt(String msg) { return false; }  // (1)
  static  boolean justDoIt(int i)      { return true; }   // (2)
}

class Athlete implements IJogger {
  public boolean justDoIt(String msg)  { return true; }   // (3)
  public boolean justDoIt(int i)       { return false; }  // (4)
}
```

```
public class RaceA {
  public static void main(String[] args) {
    Athlete athlete = new Athlete();
    IJogger jogger = athlete;
    System.out.print(jogger.justDoIt("Run"));             // (5)
    System.out.println("|" + athlete.justDoIt(10));      // (6)
  }
}
```

Select the one correct answer.

(a) The program will not compile.

(b) `true|true`

(c) `true|false`

(d) `false|true`

(e) `false|false`

Ответ: C

static vs. non-static

- ✓ static-метод поверх non-static'а – это ВСЕГДА ошибка, то есть ВО ВСЕХ случаях: интерфейс-интерфейс, класс-класс или интерфейс-класс, причем по ВСЕЙ цепочке наследования

- ✓ Когда речь идет о переопределении между классами, static и non-static-методы НИКОГДА НЕ совместимы

- ✓ Зато интерфейсный non-static-метод может «переопределить» суперинтерфейсный static-метод.
  (Это просто видимость переопределения, т.к. статические методы в интерфейсах попросту не наследуются)

- ✓ ЛЮБОЕ иное переопределение валидно

static vs. non-static

```
interface I{
//    static void run(){ System.out.println("Inter static");  }
//    default void run(){ System.out.println("Inter default");  }
    void run();
}
class Parent {                                  // ************************************
    public static void run(){                   // *   подлинный злодей живет здесь   *
        System.out.println("Parent");           // ************************************
    }
}
class Child extends Parent implements I {  // INVALID: 'overriding method is static'

//    public void run(){                        // даже закомментировали, чтобы Child не имел
//        System.out.println("Child");          // своего run()... и все равно унаследованный
//    }                                         // run() конфликтует с методом run() из I
   public static void main(String[] args) {
        new Child().run();
    }
}
```

Наследование

- ✓ Субинтерфейс может заново объявить default-метод и даже сделать его абстрактным.
- ✓ Субинтерфейс может заново объявить static-метод, но это не переопределение.
- ✓ Между интерфейсами static-метод не может переопределять abstract-метод, зато обратное допустимо (это опять чистая видимость; @Override здесь бросит комперр).
- ✓ Когда речь заходит о переменных, static может спрятать non-static, и наоборот.

Пример

```
class A {
    int x = 5;
    static int y = 7;
}

class B extends A {
    static int x = 6;
    int y = 8;
}

interface C {
    void method1();
    static void method2() {}
}

interface D extends C {
    //static void method1() {}
    void method2();
}
```

Вызовы и методы

```
interface Inter{
    static void method1(){}
    default void method2(){}
    void method3();
}
abstract class Parent implements Inter{
    public void method3(){}
}
class Child extends Parent{
    public static void main(String[] args) {
//        method1();                    // не валиден, зато Inter.method1() скомпилируется
          new Child().method2();        // VALID
          new Child().method3();        // VALID
    }
}
```

Вызовы и методы

```
class Parent {
  void one() { System.out.println("parent: one"); }
  static void two() { System.out.println("parent: two"); }
  public static void main(String[] args) {

    Parent p1 = new Parent();
    p1.one();                          // parent: one
    p1.two();                          // parent: two
//    p1.three();                      //<-- не видит

    Parent c1 = new Child();
    c1.one();                          // child: one
    c1.two();                          // parent: two
//    c1.three();                      //<-- не видит

//    Child p2 = new Parent();         // требуется каст
//    p2.one();
//    p2.two();

    Child c2 = new Child();
    c2.one();                          // child: one
    c2.two();                          // child: two
    c2.three();                        // child: three
  }
}
class Child extends Parent {
  void one() { System.out.println("child: one"); }
  static void two() { System.out.println("child: two");}
  void three() {System.out.println("child: three");}
}
```

Вызовы и методы

```
interface Parent {
    void zero();
    default void one() { System.out.println("parent: one"); }
    static void two() { System.out.println("parent: two"); }
}

class Child implements Parent {
  public void zero(){System.out.println("child: zero");}
  public void one() { System.out.println("child: one"); }
  static void two() { System.out.println("child: two"); }

  public static void main(String[] args) {
    Parent c1 = new Child();
    c1.zero();                          // child: zero
    c1.one();                           // child: one
//    c1.two();                         // этот некорректен, зато следующий скомпилируется
    Parent.two();                       // parent: two
    Child c2 = new Child();
    c1.zero();                          // child: zero
    c2.one();                           // child: one
    c2.two();                           // child: two
  }
}
```

Переменные

Допустим:

1) класс **Parent** объявил `int a = 0;`
2) класс **Child** объявил `int a = 100;`

Тогда:

```
Parent p = new Child();
System.out.println(p.a);              // 0
System.out.println(((Child)p).a);     // 100

Child c = new Child();
System.out.println(c.a);              // 100
System.out.println(((Parent)p).a);    // 0

Parent obj = new Parent();
System.out.println(obj.a);            // 0
System.out.println(((Child)obj).a);   // CCE на этапе исполнения
```

Просто супер!

```
interface Laughable {
    default void getJoke(){ System.out.println("Ha-ha-ha!"); }
    static void makeJoke(){ System.out.println("A man walks into a bar..."); }
}

interface Coder extends Laughable {
    default void sayHello() {
//        super.getJoke();                // это возможно только для классов
        Laughable.super.getJoke();
        Laughable.makeJoke();
    }
}

class JavaCoder implements Coder {
    public void sayHello() {
//        super.getJoke();                // супертип JavaCoder'а вовсе не класс
        Laughable.makeJoke();
        Coder.super.sayHello();
    }
    public static void main(String[] args) {
        new JavaCoder().sayHello();
    }
}
```

Вопросы?

Приведение типов

Кастинг

```
Primate primate = new Lemur();  // Implicit Cast
Lemur lemur2 = primate;         // DOES NOT COMPILE
System.out.println(lemur2.age);

Lemur lemur3 = (Lemur)primate;  // Explicit Cast
System.out.println(lemur3.age);
```

Правила

1. Casting a reference from a subtype to a supertype doesn't require an explicit cast.
2. Casting a reference from a supertype to a subtype requires an explicit cast.
3. The compiler disallows casts to an unrelated class.
4. At runtime, an invalid cast of a reference to an unrelated type results in a `ClassCastException` being thrown.

Неродственные типы

```
public class Bird {}

public class Fish {
  public static void main(String[] args) {
    Fish fish = new Fish();
    Bird bird = (Bird)fish;  // DOES NOT COMPILE
  }
}
```

ClassCastException

```
public class Rodent {}

public class Capybara extends Rodent {
   public static void main(String[] args) {
      Rodent rodent = new Rodent();
      Capybara capybara = (Capybara)rodent;  // ClassCastException
   }
}
```

instanceof

```
if(rodent instanceof Capybara) {
    Capybara capybara = (Capybara)rodent;
}
```

Неродственные типы

```
public static void main(String[] args) {
   Fish fish = new Fish();
   if (fish instanceof Bird) {  // DOES NOT COMPILE
      Bird bird = (Bird) fish;  // DOES NOT COMPILE
   }
}
```

Два потомка

```
public class XYZ {
    public static void main(String[] args) {
        X z = new Z();
        X y = new Y();
        Z y1 = (Z) y; // y - переменная класса X, поэтому cast проходит компилятор
    }
}

class X {}
class Y extends X {}
class Z extends X {}
```

Пример

```
class Light { /* … */ }
class LightBulb extends Light { /* … */ }
class SpotLightBulb extends LightBulb { /* … */ }
class TubeLight extends Light { /* … */ }
class NeonLight extends TubeLight { /* … */ }

public class WhoAmI {
  public static void main(String[] args) {
    boolean result1, result2, result3;
    Light light1 = new LightBulb();                    // (1)
    //  String str = (String) light1;                  // (2) Compile-time
error!
    //  result1 = light1 instanceof String;            // (3) Compile-time
error!

    result2 = light1 instanceof TubeLight;             // (4) false: peer
class.
    //  TubeLight tubeLight1 = (TubeLight) light1;     // (5)
ClassCastException!

    result3 = light1 instanceof SpotLightBulb;         // (6) false:
superclass.
    //  SpotLightBulb spotRef = (SpotLightBulb) light1;// (7)
ClassCastException!

    light1 = new NeonLight();                          // (8)
    if (light1 instanceof TubeLight) {                 // (9) true.
      TubeLight tubeLight2 = (TubeLight) light1;       // (10) OK.
      // Can now use tubeLight2 to access an object of the class NeonLight,
      // but only those members that the object inherits or overrides
      // from the class TubeLight.
    }
  }
}
```

Упаковка

```
// (1) Boxing and casting: Number <- Integer <- int:
Number num = (Number) 100;
// (2) Casting, boxing, casting: Object <- Integer <- int <- double:
Object obj = (Object) (int) 10.5;
// (3) Casting, unboxing, casting: double <- int <- Integer <- Object:
double d = (double) (Integer) obj;
```

Добраться до…

```
class Casting{
    public static void main(String[] args) {
        Object obj = new StringBuilder("Cast me plz... ");
//        obj.append("Done!");                                  // INVALID
        (  (StringBuilder)obj  ).append("Done!");
        System.out.println(obj);
    }
}
```

Герои идут в обход

```
class A {
    public String str = "Hello from A!";
}

class B extends A {
    private String str = "Hello from B!";
}

class C extends B{}

class Test{
   public static void main(String args[]){
      C c = new C();
//       System.out.println(c.str);                    // INVALID
        System.out.println( ((A)c).str );              // Hello from A!
   }
}
```

Интерфейсы

```
interface I1 {}
interface I2 {}
class Test2 {                                   // Test нефинален, иначе был бы комперр
    public static void main(String[] args) {
        Test2 t = new Test2();                  // кстати, t можно сделать финальной
        I1 i1 = (I1) t;                         // класс - интерфейс; бросает CCE
        I2 i2 = (I2) i1;                        // интерфейс - интерфейс; бросает CCE
    }
}
```

Классы и интерфейсы

```
public class Start {
    public static void main(String[] args) {
        Car car = new FCar();
        ((Breakable) car).breakS(); // Компилируется, хотя интерфейс никак не родственник классу Car.
                                    // В runtime полетит ClassCastException
        //((String) car).length();  // Не компилируется, потому что классы не родственники
    }
}

interface Breakable {
    public void breakS();
}

class Car {}

class FCar extends Car {}
```

instanceof

```
Number tickets = 4;
if(tickets instanceof String) {}  // DOES NOT COMPILE

Number tickets = 5;
if(tickets instanceof List) {}
```

```
public class MyNumber extends Number implements List

Integer tickets = 6;
if(tickets instanceof List) {}  // DOES NOT COMPILE
```

Упражнение

```
class Base {
    public static void main(String[] args) {
        Derived d = new Derived();
        Base b = new Base();

        // line X: insert code here
    }
}

class Derived extends Base { }
interface I{}
```

Which LOC, when inserted at line X, throws a ClassCastException?

A. d = b;
B. b = d;
C. d = (Object) b;
D. d = (I) d;
E. d = (Derived)(I) b;

Ответ: E

Разбор упражнения

✓ Любой явный каст бросает CCE,
кроме случаев, когда актайп IS-A кастайп (идем справа налево)

```
interface I{}
class A implements I{}
class B extends A{}
class C extends B{}

class Casting{
    public static void main(String[] args) {
        A a = new A();
        B b = new B();
        C c = new C();

// *** здесь CCE нет ***
        a = (A)(I) b;
        a = (B)(I) b;
        b = (B)(A)(I) c;
// **********************

// раскомментируем следующие строки по очереди, но перед этим
// надо закомментировать предшествующую секцию (там, где нет CCE)
//        b = (B)(I) a;      // CCE @ A -> B
//        a = (I) b;         // INVALID
//        I i = (C) a;       // CCE @ A -> C

//        A huh = (B)(I)(B)(A)(C)(B)(A)(I) b;     // CCE @ B -> C
//        A huh = (B)(I)(B)(A)   (B)(A)(I) b;     // на сей раз никакого CCE
    }
}
```

Упражнение

```
 1 class Jogger {
 2     public static void main(String[] args) {
 3
 4         Jogger jogger = new Runner();
 5         FitnessBuff runner = new Runner();
 6
 7         jogger.move();
 8         (FitnessBuff)jogger.move();
 9         ((FitnessBuff)jogger).move();
10         runner.move();
11         (FitnessBuff)runner.move();
12         ((FitnessBuff)runner).move();
13     }
14 }
15
16 class Runner extends Jogger implements FitnessBuff {
17     public void move() { System.out.println("Make way!"); }
18 }
19 interface FitnessBuff { public void move(); }
```

Which three LOCs compile and output Make way!**?**

A. Line 7
B. Line 8
C. Line 9
D. Line 10
E. Line 11
F. Line 12

Ответ: CDF

instanceof

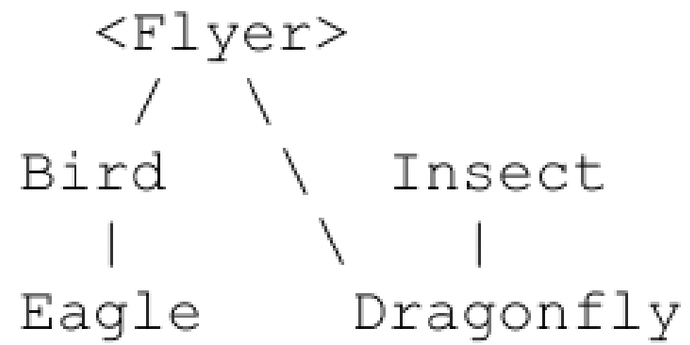
- ✓ instanceof рассчитан только на ссылочные типы; попытка использовать его с примитивами дает комперр;
- ✓ instanceof бросает комперр, если левый операнд нельзя преобразовать к правому;
- ✓ за исключением случаев, когда класс является final, ни его каст к чуждому интерфейсному типу, ни попытка использовать такой интерфейсный тип в качестве операнда instanceof не бросит комперр;
- ✓ instanceof всегда дает false, кроме случаев, когда его левый операнд не имеет значение null и может быть скастирован к правому операнду без выброса CCE.

Пример

```
interface Flyer{ }

class Bird implements Flyer { }
class Eagle extends Bird { }

class Insect { }
class Dragonfly extends Insect implements Flyer {}
```




```
class Test {
    public static void main(String[] args) {
        Flyer fe = new Eagle();
        Eagle ee = new Eagle();
        Insect id = new Dragonfly();
        Insect ii = new Insect();

        if(fe instanceof Flyer) System.out.println("fe is a Flyer");  // строка t1
        if(ee instanceof Bird) System.out.println("ee is a Bird");    // строка t2
//        if(id instanceof Bird) System.out.println("id is a Bird");  // строка t3
        if(id instanceof Flyer) System.out.println("id is a Flyer");  // строка t4
        if(ii instanceof Flyer) System.out.println("ii is a Flyer");  // строка t5
    }
}
```

Упражнение

```
interface I { }

public class Parent implements I {
    public String toString() { return "P "; }
    public static void main(String[] args) {
        Child c = new Child();
        Parent p = c;
        I inter = p;
        System.out.print(p);
        System.out.print((Child)p);
        System.out.print(inter);
    }
}

class Child extends Parent {
    public String toString() { return "C "; }
}
```

What is the result?

A. P P P
B. P C P
C. C C P
D. C C C
E. Compilation fails

Ответ: D

Упражнение

```
public class MyClass {
  public static void main(String[] args) {
    A[] arrA;
    B[] arrB;

    arrA = new A[10];
    arrB = new B[20];
    arrA = arrB;        // (1)
    arrB = (B[]) arrA; // (2)
    arrA = new A[10];
    arrB = (B[]) arrA; // (3)
  }
}
```

```
class A {}

class B extends A {}
```

Select the one correct answer.

(a) The program will fail to compile because of the assignment at (1).

(b) When run, the program will throw a `java.lang.ClassCastException` in the assignment at (2).

(c) When run, the program will throw a `java.lang.ClassCastException` in the assignment at (3).

(d) The program will compile and run without errors, even if the cast operator `(B[])` in the statements at (2) and (3) is removed.

(e) The program will compile and run without errors, but will not do so if the cast operator `(B[])` in statements at (2) and (3) is removed.

Ответ: C

Неоднозначная ссылка

```
interface I1{ int A = 10; }
interface I2{ int A = 20; }

class Test implements I1, I2{
    public static void main(String[] args) {
//        System.out.println( new Test().A );          // INVALID: "reference
                                                       // to A is ambiguous"
        System.out.println(  I1.A  );                  // печатает 10
        System.out.println( ( (I1)new Test() ).A );    // печатает 10
        System.out.println( ( (I2)new Test() ).A );    // печатает 20
    }
}
```

Вопросы?

Пространства имён

Загораживание

- ✓ JLS: A simple name may occur in contexts where it may potentially be interpreted as the name of a variable, a type, or a package. In these situations, the rules of §6.5 (JLS) specify that a variable will be chosen in preference to a type, and that a type will be chosen in preference to a package. Thus, it is may sometimes be impossible to refer to a visible type or package declaration via its simple name. We say that such a declaration is obscured.

- ✓ Типичный случай загораживания – это когда совпадают имена дата-типов в разных пакетах.

Пространства имён

```
public class Name {

  Name() {                          // (1) No-argument constructor
    System.out.println("Constructor");
  }

  void Name() {                     // (2) Instance method
    System.out.println("Method");
  }

  public static void main(String[] args) {
    new Name().Name();              // (3) Constructor call followed by method
call
  }
}
```

Output from the program:

```
Constructor
Method
```

Пространства имён

- ✓ В Java определены четыре пространства имен (namespaces): для полей, для методов, для дата-типов и, наконец, для пакетов.

String String String

```
class String{
    String(){}
    String(String String){}
    String String(){ return new String(); }
    public static void main(String[] string) {
        String String = new String(new String().String());
        System.out.println("String!");
    }
}
```

String String String

```
class String{
    String(){}
    String(String String){}
    String String(){ return new String(); }
    public static void main(String[] string) {
        String String = new String(new String().String());
        System.out.println("String!");
    }
}
```

```
                    public static void main(java.lang.String[] string) {
```

Вопросы?

Домашнее задание

Тест

Пожалуйста, пройдите опрос

https://otus.ru/polls/17836/

Спасибо за внимание!

Кастуйте и не загораживайтесь!