1360 users solved this topic. Latest completion was about 3 hours ago.

Theory: Runtime type checking

28 minutes 0 / 5 problems solved

Skip this topic Start practicing

§1. Runtime type checking

A variable of a base class can always refer to an object of a subclass. We can determine the actual type of the referred object at runtime.

Java provides several ways to do it:

- the `instanceof` operator that can be used for testing if an object is of a specified type;
- **java reflection** that can be used to obtain an object representing the class.

Let's consider these ways to check types of objects at runtime.

Here is a class hierarchy which we will use in examples:

```
class Shape {...}

class Circle extends Shape {...}

class Rectangle extends Shape {...}
```

The hierarchy is very simple, the fields and methods of classes are hidden for clarity. However, this hierarchy demonstrates the "IS-A" relation pretty well.

§2. The keyword instanceof

The binary operator `instanceof` returns `true` if an object is an instance of a particular class or its subclass.

The base syntax is the following:

```
obj instanceof Class
```

We've created a couple of instances of the classes above:

```
Shape circle = new Circle();  // the reference is Shape, the object is Circle
Shape rect = new Rectangle(); // the reference is Shape, the object is Rectangle
```

Let's determine their types:

```
boolean circleIsCircle = circle instanceof Circle; // true
boolean circleIsRectangle = circle instanceof Rectangle; // false
boolean circleIsShape = circle instanceof Shape; // true

boolean rectIsRectangle = rect instanceof Rectangle; // true
boolean rectIsCircle = rect instanceof Circle; // false
boolean rectIsShape = rect instanceof Shape; // true
```

So, the `instanceof` operator allows you to determine the actual type of an object even if it is referred to by its superclass.

As you can see, this operator considers a subclass object an instance of the superclass:

```
boolean circleIsShape = circle instanceof Shape; // true
```

Pay attention, the type of the object in question should be a subtype (or the type) of the specified class. Otherwise, the statement cannot be compiled.

Here is a non-compiled example:

Current topic:

Runtime type checking ···

Topic depends on:

✕ Referencing subclass objects ···

Topic is required for:

hashCode() and equals() ···

Table of contents:

```
Circle c = new Circle();
boolean circleIsRect = c instanceof Rectangle; // Inconvertible types
```

The second line contains the compile-time error: Inconvertible types.

§3. Use reflection

Each object has a method `getClass` that can be used to obtain an object representing the class. We can directly compare the classes represented by objects at runtime using java reflection.

Let's consider an example. Here is an instance of `Circle`:

```
Shape circle = new Circle();
```

Let's test it using reflection:

```
boolean equalsCircle = circle.getClass() == Circle.class; // true
boolean equalsShape = circle.getClass() == Shape.class;   // false
boolean rectangle = circle.getClass() == Rectangle.class; // false
```

Unlike the `instanceof` operator, this approach performs strict type testing and does not see subclass objects as instances of the superclass.

There is also another way to check types. An object representing the class has a method `isInstance` that is similar to the `instanceof` keyword.

Let's test the object `circle` again.

```
boolean isInstanceOfCircle = Circle.class.isInstance(circle); // true
boolean isInstanceOfShape = Shape.class.isInstance(circle); // true

boolean isInstanceOfRectangle = Rectangle.class.isInstance(circle); // false
```

Similar to the `instanceof` operator, this method considers a subclass object as an instance of its superclass. However, unlike the operator, the following example is successfully compiled:

```
Circle c = new Circle();
boolean circleIsRect = Rectangle.class.isInstance(c); // false
```

You can use any of the described approaches to determine the actual type of the referred object.

§4. When to use it

If you cast a superclass object to its subclass, you may get a `ClassCastException` if the object has another type. Before casting, you can check the actual type using one of the approaches we've considered in this topic.

The following example demonstrates it.

```
Shape shape = new Circle();

if (shape.getClass() == Circle.class) {
    Circle circle = (Circle) shape;

    // now we can process it as a circle
}
```

Keep in mind, a lot of runtime checks in the program may indicate a poor design. Use runtime **polymorphism** to reduce them.

Report a typo

113 users liked this theory. 1 didn't like it. **What about you?**

😍 🙂 😐 🙁 😡

Start practicing

Comments (7) Hints (0) Useful links (0) Show discussion