Theory: Objects

13 minutes 0 / 4 problems solved

Skip this topic Start practicing

7595 users solved this topic. Latest completion was about 1 hour ago.

A typical object-oriented program consists of a set of interacting **objects**. Each object has its own state separated from others. Each object is an instance of a particular class (type) that defines common properties and possible behavior for its objects.

All classes from the standard library (`String`, `Date`) and classes defined by programmers are **reference types** which means that variables of these types store addresses where the actual objects are located. In this regard, the comparison and assignment operations work with objects differently than with primitive types.

§1. Creating objects

The keyword **new** creates an object of a particular class. Here we create a standard string and assign it to the variable `str`:

```
String str = new String("hello");
```

The variable `str` stores a reference to the object **"hello"** located somewhere in the heap memory.

In the same way, we can create an object of any class we know.

Here is a class that describes a patient in a hospital information system:

```
class Patient {
    String name;
    int age;
}
```

Here is an instance of this class:

```
Patient patient = new Patient();
```

Despite the fact that `String` is a standard class and `Patient` is our own class, both classes are regular reference types. However, there is a big difference between those classes and we will discuss it below.

§2. Immutability of objects

There is an important concept in programming called **immutability**. Immutability means that an object always stores the same values. If we need to modify these values, we should create a new object. The common example is the standard `String` class. Strings are immutable objects so all string operations produce a new string. Immutable types allow you to write programs with fewer errors.

The class `Patient` is not immutable because it is possible to change any field of an object.

```
Patient patient = new Patient();

patient.name = "Mary";
patient.name = "Alice";
```

In the following topics, we will look at the existing immutable classes as well as learn how to create new ones and when to use them.

§3. Sharing references

More than one variable can refer to the same object.

Current topic:

Objects Stage 3

Topic depends on:

✓ Primitive and reference types Stage 2

✓ Defining classes Stage 3

Topic is required for:

Static members

Patterns and Matcher

BigInteger

Random Stage 5

✓ StringBuilder Stage 3

LocalDate

LocalTime

Boxing and unboxing

Inside the JVM

Garbage Collector

Threads as objects

Class files and Bytecode

Files

Table of contents:

```java
Patient patient = new Patient();

patient.name = "Mary";
patient.age = 24;

System.out.println(patient.name + " " + patient.age); // Mary 24

Patient p = patient;


System.out.println(p.name + " " + p.age); // Mary 24
```

It is important to understand that two variables refer to the same data in memory rather than two independent copies. Since our class is mutable, we can modify the object using both references.

```java
patient.age = 25;
System.out.println(p.age); // 25
```

§4. Nullability

As for any reference types, a variable of a class-type can be **null** which means it is not initialized yet.

```java
Patient patient = null;
```

This is a common feature in Java available for classes since they are reference types.

§5. Conclusion

By now, not only have we already worked with some classes from the standard library but also learned how Java allows us to create our own classes. In this topic, we've discussed that the nature of custom classes' objects and standard library ones are based on the same principles.

Keep in mind, that classes defined by programmers are **reference types**. When objects are created by the **new** operator it returns reference in memory where the created objects are located. By this reference, we can get access to its fields and change them. Several variables can refer to the same object through a reference. It is also possible to create two independent objects with the same field's content. It's important to understand that references to such objects are different. However, not all objects allow changing its state after creation. Such a feature is called **immutability**.

Report a typo

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

Start practicing

Comments (5) Hints (0) Useful links (1) Show discussion