JAVA SEMINAR

DAY 07 - GENERICS

JAVA SEMINAR

Contents

Let's dwelve deeper into OOP by studying one of Java's specificities: the generics.

Generics allow a class or a method to support multiple type while keeping the compile-time type safety. Without knowing it, you already used generics in day 01 by using the `ArrayList` type.

Let's have a look at the following block of Java code:

```
List v = new ArrayList();
v.add("test");
Integer i = (Integer) v.get(0); // runtime error
```

Although the code compiles without error, it throws a runtime exception `java.lang.ClassCastException` when executing the third line.

Take some time to understand this example! Test it on your own machine.

To avoid bad surprises, we'd like to detect this error at compilation time, not at runtime.
To cope with this, we need to use generics. In fact, this is the primary motivation for generics.

Exercise 01

Delivery: `./Solo.java`

Create a generic class named **Solo** that holds a value.

The class must have as attribute the value to hold, named `value`.

This attribute must have a getter (`getValue`) and a setter (`setValue`).

The constructor of this class takes the `value` as parameter.

You probably need to make some research on Internet about generics' syntax...

We should be able to use your class strictly like in the following example:

```
Solo<String> strSolo = new Solo<>("toto");
String strValue = strSolo.getValue();
strSolo.setValue("tata");

Solo<Integer> intSolo = new Solo<>(Integer.valueOf(42));
Integer intValue = intSolo.getValue();
intSolo.setValue(Integer.valueOf(1337));
```

Exercise 02

Delivery: `./Pair.java`

Create a `Pair` generic class, that contains a pair of elements.

The types of both elements are a priori undefined.

The class must have the following attributes and one method:

- ✓ `first` is the first element of the pair, its type being mentioned as `T` ;
- ✓ `second` is the second element of the pair, its type being mentioned as `V` ;
- ✓ the `display` method that pretty prints the pair like this `first: [first], second: [second].`

The attributes need a getter (`getFirst`, `getSecond`) but no setter.

The constructor of this class takes respectively the first and second element as parameters.

Exercise 03

Delivery: `./Duet.java`

Create a `Duet` class that has two public static generic methods (that’s long to say!) `min` and `max`.

These methods should take two `T` type parameters and compare them using the `compareTo` method.

The `min` method returns the smallest one, whereas the `max` method returns the highest one.

smallest and *highest* can apply to many different types (not only numerical values, according to the `compareTo` method, which defines the ordering relation to be used.)

To use the `compareTo` method, ensure that `T` extends from the **Comparable interface**.

Exercise 04

Delivery: ./Character.java, ./Warrior.java, ./Mage.java, ./Movable.java, ./Solo.java, ./Pair.java, ./Duet.java, ./Battalion.java

Okay, now that you've started to understand how generics work, we're going to create battalions composed of `Warrior` and `Mage` (and potentially any other type of `Character`).

First, copy the classes you wrote the previous day. Then, create a `Battalion` class that has:

- ✓ one `characters` attribute of type List. It holds all characters composing the battalion.
- ✓ a `add` public method which takes a List of Character objects (or any other object that inherits from Character) and add them to the battalion.
- ✓ a `display` public method which displays the name of every character in `characters`.

For instance:

```java
public static void main(String args[]) {
    List<Mage> mages = new ArrayList<>();
    mages.add(new Mage("Merlin"));
    mages.add(new Mage("Mandrake"));
    List<Warrior> warriors = new ArrayList<>();
    warriors.add(new Warrior("Spartacus"));
    warriors.add(new Warrior("Clovis"));
    Battalion battalion = new Battalion();
    battalion.add(mages);
    battalion.add(warriors);
    battalion.display();
}
```

Terminal

```
T-JAV-500> java Example
Merlin: May the gods be with me.
Mandrake: May the gods be with me.
Spartacus: My name will go down in history!
Clovis: My name will go down in history!
Merlin
Mandrake
Spartacus
Clovis
```

Exercise 05

Delivery: `./Character.java`, `./Warrior.java`, `./Mage.java`, `./Movable.java`, `./Solo.java`, `./Pair.java`, `./Duet.java`

Add the integer attribute `capacity` to the `Character` class. Set it to 0 by default.
Add a new constructor taking the same parameters as the existing one but adding the capacity.

For Warriors, this parameter represents it's power.
For Mages, this parameter represents it's magnetism.

The Character class should also now implement the **Comparable interface**.

The implemented `compareTo` method will follow the following rules:

- ✓ if the argument is another character:
  - if characters are both of the same type, compare the capacities ;
  - if there is a Warrior and a Mage, the Mage is the greatest unless the Warrior's capacity is a multiple of the Mage's capacity (in which case, the Warrior is the greatest) ;
- ✓ else just return 0.

You need to be smart to compare a Mage and a Warrior...

For instance:

```
Character merlin   = new Mage("Merlin", 12);
Character gandalf   = new Mage("Gandalf", 12);
Character mandrake = new Mage("Mandrake", 9);
Character achilles = new Warrior("Achilles", 240);
merlin.compareTo(mandrake); // Should return a positive value
merlin.compareTo(achilles); // Should return a negative value
gandalf.compareTo(merlin); // Should return 0
```

Exercise 06

Delivery: `./Character.java`, `./Warrior.java`, `./Mage.java`, `./Movable.java`, `./Solo.java`, `./Battalion.java`, `./Pair.java`, `./Duet.java`

Add a **fight** method to your `Battalion` class.
It will take the first two `characters` of the battalion and make them battle against each other.

The winner will be determined using it's `compareTo` method.

The loser of the fight is then removed from the battalion.
He is not worthy enough to stay... if he's still alive!

The `fight` method returns:

- ✓ `true` if there is a fight ;
- ✓ `false` is there is none (no characters in the battalion, ...).

In case of a tie, both characters are removed from the battalions.
They're not worthy if they can't defeat their opponents.

v 3.4

8