Detecting annotations

The Spring library has lots of them, and there's a useful @Test annotation for testing Java classes.

getDeclaredAnnotations method

Actually, annotations are just markers: they indicate that some class, method, or field has special properties.

```
class Item {
    @Deprecated
    public static final int maxItems = 100;
    public static int inStock = 19;

    private String name;
    protected int basePrice;

    public Item(String name, int basePrice) {
        this.name = name;
        this.basePrice = basePrice;
    }

    public String getName() {
        return name;
    }

    public int getPrice() {
        return (int) (basePrice * getMarkUp());
    }

    public boolean haveSuchAmount(@Deprecated int amount) {
        return inStock >= amount;
    }

    @Deprecated
    protected double getMarkUp() {
        double markUp = 0.1;
        // ... connecting to the remote server
        return 1 + markUp;
    }

    @Override
    public String toString() {
        return getName() + " " + getPrice();
    }
}
```

It has three @Deprecated annotations and one @Override annotation. Let's try to analyze them by using reflection.
For working with annotations, Class, Method, Field and Constructor classes have the getDeclaredAnnotations() method, that returns an array of Annotation objects.

The Annotation class provides a useful method called annotationType that returns a Class object representing the annotation. It can be useful if you need to access all elements of the annotation or just get its name.

Code example

Let's execute the following code. Here we go through all the names of methods and fields of our Item class and apply the getDeclaredAnnotations() method to get their annotations and print them next to the respective names:

```
Class itemClass = Item.class;

for (Field field : itemClass.getDeclaredFields()) {
   for (Annotation annotation : field.getDeclaredAnnotations()) {
      System.out.print(annotation + " - ");
   }
   System.out.println(field.getName());
}

for (Method method : itemClass.getDeclaredMethods()) {
   for (Annotation annotation : method.getDeclaredAnnotations()) {
      System.out.print(annotation + " - ");
   }
   System.out.println(method.getName());
}
```

Take a look at the output:

```
@java.lang.Deprecated(forRemoval=false, since="") - maxItems
inStock
name
basePrice
toString
getName
@java.lang.Deprecated(forRemoval=false, since="") - getMarkUp
getPrice
haveSuchAmount
```

However, for some reason, there is no @Override toString line in here.

We may suppose that the absence of @Override has to do with its method: actually, toString is not declared in this class, but in the Object.
To check our hypotheses let’s use another method for getting class methods getMethods. As you probably remember, it returns all public methods of the class, including the inherited ones.

```
for (Method method : itemClass.getMethods()) {
   for (Annotation annotation : method.getDeclaredAnnotations()) {
      System.out.print(annotation + " - ");
   }
   System.out.println(method.getName());
}
```

If our conjecture is correct, this time we will get the @Override annotation for toString method. But look what we get by the above snippet:

```
toString
getName
getPrice
haveSuchAmount
wait
wait
wait
equals
@jdk.internal.HotSpotIntrinsicCandidate() - hashCode
@jdk.internal.HotSpotIntrinsicCandidate() - getClass
@jdk.internal.HotSpotIntrinsicCandidate() - notify
@jdk.internal.HotSpotIntrinsicCandidate() - notifyAll
```

Still, something is not right! As you see, we now got four methods with unfamiliar annotations, and the toString method still has no annotation.
Note that this time we didn’t get getMarkUp method at all since it is *protected*.

Hence we may conclude that the reason lies in the nature of the annotation itself.

Indeed, @Override annotation is a compile-time annotation and is not present when the program is running. While @Deprecated is present during the execution of the program, which means that it is a runtime annotation.

Conclusion
Reflection package provides functionality to retrieve annotations from the source code. Class, Field and Method classes have the getDeclaredAnnotations method, which returns a list of configured runtime annotations. Other annotations are not available at runtime

because the compiler erases them.

In Java, the Annotation interface is part of the java.lang.annotation package. This package contains the classes and interfaces related to Java's annotation mechanism.
(It is not part of reflect)

```
import java.lang.annotation.Annotation;

class AnnotationUtils {

    public static void printClassAnnotations(Class classObject) {
        // write your code here
        for(Annotation ann : classObject.getDeclaredAnnotations()){
            Class c =  ann.annotationType();
            String s = c.getName();
            System.out.println(s.substring(s.lastIndexOf(".") + 1));
        }
    }
}
```