PF3: Assignment 1

Due on October 8, 2018 at 8:30am

Prof. Walter Binder

A. Romanelli

Problem 1

1 .1 `"This is printed in class B"`, it comes from the method print defined in B, which overloads the original method defined in A, but since the type of the argument B, thus the latter is called.

.2 An abstract class cannot be instantiated: `new A()` is an invalid statement.

.3 Argument mismatch: a method that takes a String as type of the argument cannot be found and thus an exception is thrown.

.4 `"This is printed in class A"`, it comes from the method print defined in A, after an object of class B is casted into an object of class A. This means that the object of class B on which the method is called as both methods for an argument of type A and an argument of type B, since the argument is of type A, the method inherited from class A is used.

.5 `"This is printed in class A"` is the result. The object a is instantiated from the class B and that's its dynamic type, but it's declared type is A and thus in the method call, the argument type is going to be A and thus the method defined in A will be called.

.6 The method print is not defined within the class Object and would thus throw an exception.

.7 Abstract class A cannot be instantiated.

.8 `"This is printed in class A"`. This happens because we instantiated an object of class B but cast it into an object of type A, thus not having the method defined in the B class. When we call the method print passing an object of class B, the method A is used anyway because it's the only one available and accepts all subtypes of class A.

.9 Argument mismatch, a method that takes a String as argument cannot be found and thus raises an exception.

.10 Abstract class A cannot be instantiated.

2 .1 Overload, the two methods coexist and both can be used depending on the type of the provided argument.

.2 Override, the two methods share the same argument type, hence when the method print is defined again in C (subclass of A) taking the same argument type, it overrides the previously defined method with the new one.

.3 None, because B and D are not related besides having a common parent of A and thus their methods do not have any interaction.

.4 Override, the method defined in class D was already defined in class C from which it is inheriting and having the same argument type, the latter will override the former.

.5 None, since the method defined in C is private, D does not inherit it and when it attempts to define the same method in class D, it does not override nor overload anything.

3 .1 The declared type is Object and the possible dynamic types of ∘ are: Object/A/B/C/D;

.2 The declared type is D and ∘ accepts only D as dynamic type;

.3 Declared type C and ∘ accepts C and D as dynamic types;

.4 Declared type A and accepts all its subtypes as dynamic types: A, B, C, D;

.5 Declared type B, accepting as dynamic types: B.

Problem 2

1 `IntSet2` violates the SP because it may throw `DuplicateNotAllowedException`, which the superclass method does not declare and thus this will cause an error at compile time.

2 No, it does not violate the SP, `int` is converted into an Integer through auto-boxing and `IntSet2.insert` overloads `IntList.insert` without changing return type, parameters types or thrown exceptions.

Problem 3

```
1  import java.math.BigDecimal;
2
3  class BankAccount {
4
5      private BigDecimal amount;
6
7      public BankAccount(BigDecimal amount) {
8          this.amount = amount;
9      }
10
11     public void transferTo(BankAccount target, BigDecimal amount) {
12         // Local variables for Postconditions check
13         BigDecimal initialAmount = this.amount;
14         BigDecimal initialTargetAmount = target.amount;
15         // Invariants (2)
16         assert target.amount.compareTo(BigDecimal.ZERO) != -1: "The target BankAccount must
       have a positive amount";
17         assert this.amount.compareTo(BigDecimal.ZERO) != -1: "The current BankAccount must
       have a positive amount";
18         // Preconditions (3)
19         assert ((amount.compareTo(BigDecimal.ZERO)) != -1) && (amount.scale() == 2): "Amount
        to transfer must be positive and have two digits";
20         assert this != target: "Transfer target cannot be the same as the origin";
21         assert this.amount.compareTo(amount) != -1: "Cannot transfer more money that the
       BankAccount has";
22         // Code
23         this.amount = this.amount.subtract(amount);
24         target.amount = target.amount.add(amount);
25         // Invariants (2)
26         assert target.amount.compareTo(BigDecimal.ZERO) != -1: "The target BankAccount must
       have a positive amount";
27         assert this.amount.compareTo(BigDecimal.ZERO) != -1: "The current BankAccount must
       have a positive amount";
28         // Postconditions (2)
29         assert this.amount.add(amount).compareTo(initialAmount) == 0: "The money we
       transfered + the money we have must be equal to the money we originally had";
30         assert target.amount.subtract(amount).compareTo(initialTargetAmount) == 0: "The
       money the target has - the money we transfered must be equal to the money it had";
31     }
32
33     public String toString() {
34         return amount.toString();
35     }
36 }
```

Problem 4

```
class A {
    public String show(Object obj) {
        return ""+this.getClass().getSimpleName()+".show("+obj.getClass().getSimpleName()+")
    ";
    }
}

class B extends A {}

class C extends B {}

class D extends C {}

public class Main {
    public static void main(String[] args) {
        A a1 = new A();
        A a2 = new B();
        B b = new B();
        C c = new C();
        D d = new D();
        System.out.println(a1.show(b)); // 1
        System.out.println(a1.show(c)); // 2
        System.out.println(a1.show(d)); // 3
        System.out.println(a2.show(b)); // 4
        System.out.println(((B) a2).show(b)); // 5
        System.out.println(a2.show(c)); // 6
        System.out.println(a2.show(d)); // 7
        System.out.println(b.show(b)); // 8
        System.out.println(b.show(c)); // 9
        System.out.println(b.show(d)); // 10
    }
}
```

This implementation of the A class provides the exact same functionality of initially proposed design.