Università della Svizzera italiana **Advanced Java Programming**
Assignment 1 **Fall 2022**
Dr. Andrea Rosà, Prof. Walter Binder

Instructions

- This assignment consists of 20 points and counts 2% towards your overall grade.
- The deadline of the assignment will be published on iCorsi. The deadline is strict.
- You are supposed to submit a pdf containing your (textual) solutions on iCorsi. Use the following pattern for naming your submission files: $< firstname > . < lastname > .pdf$.
- The assignment considers (vanilla) Java version 17. For simplicity, code snippets may not always contain the full code (e.g. imports or called methods might be omitted). If not mentioned otherwise, you can assume that the snippets compile, the hidden code is implemented correctly, and methods do what their name suggests, without throwing any runtime exceptions.

Exercise A: Foundations of Java I (10 points)

Please answer the questions below (single choice). For correct answers, you get one point. Otherwise, one is subtracted. You get at least 0 points for the complete exercise.
Consider the following code snippet:

```
1 package ch.zoo;
2 class Tiger {
3     private static final Number NUMBER_OF_TEETH = new Integer(30);
4     protected final String name;
5     public Tiger(String name) { this.name = name; }
6     public boolean equals(Object x){/* Code hidden */}
7 }
```

Which of the following statements are true?

1. `Tiger` extends `Object`
   ☐ True ☐ False
2. Given that `equals(Object x)` is overriden, a `compareTo(Object x)` implementation should be added.
   ☐ True ☐ False
3. Given that `equals(Object x)` is overriden, a `hashCode()` implementation should be added.
   ☐ True ☐ False
4. `equals(Object x)` is used when evaluating for equality through the `==` operator.
   ☐ True ☐ False

5. The `equals(Object x)` method is implemented by default in every class and should thus not be overridden in `Tiger`.
   ☐ True ☐ False

6. Assume a method `void makeMedicine(Tiger t);` in another class. t is guaranteed to be an immutable object.
   ☐ True ☐ False

7. `Tiger` is callable from any other package, as it has a public constructor.
   ☐ True ☐ False

8. Only classes extending `Tiger` can read the value of the field `name`.
   ☐ True ☐ False

9. Classes extending `Tiger` must explicitly implement at least one constructor.
   ☐ True ☐ False

10. The declared type of `NUMBER_OF_TEETH` is `INTEGER`, the dynamic type is `int`.
    ☐ True ☐ False

Exercise B: Foundations of Java II (5 points)

Please answer the questions below (single choice). For correct answers, you get one point. Otherwise, one is subtracted. You get at least 0 points for the complete exercise.
Now consider the well known `java.util` collections.

1. You can create a collection of (unboxed) primitive types
   ☐ True ☐ False

2. `HashMap`s may contain duplicate keys if keys are mutable.
   Hint: Consider the case where you mutate a key present in the map and call `put(..)` again with the same key instance.
   ☐ True ☐ False

The following statements are independent of the collections.

3. Consider two Strings a and b. `a == b` implies `a.equals(b)`.
   ☐ True ☐ False

4. You can throw runtime exceptions anywhere and the code will still compile.
   ☐ True ☐ False

5. After a try-catch block, `finally` blocks are evaluated if and only if no exception is caught.
   ☐ True ☐ False

Exercise C: Polymorphism (5 Points)

Please answer the questions below (single choice). For correct answers, you get one point. Otherwise, one is subtracted. You get at least 0 points for the complete exercise.
Consider the following snippet:

```
1  class A {
2      String getName(){ return "I'm A"; }
3
4      void printSomething(A aParam){
5          System.out.println("A or B at A: " + aParam.getName());
6      }
7  }
8
9  class B extends A {
10     String getName(){ return "I'm B"; }
11
12     void printSomething(A aParam){
13         System.out.println("A at B: " + aParam.getName());
14     }
15
16     void printSomething(B bParam){
17         System.out.println("B at B: " + bParam.getName());
18     }
19 }
20
21 public class Polymorph {
22     public static void main(String[] args) {
23         A a = new A();
24         B b = new B();
25
26         // Next lines output: A or B at A: I'm A
27         a.printSomething(a);
28         // Next lines output: A or B at A: I'm B
29         a.printSomething(b);
30         // Next lines output: A at B: I'm A
31         b.printSomething(a);
32         // Next lines output: B at B: I'm B
33         b.printSomething(b);
34     }
35 }
```

1. `printSomething(A aParam)` on line 12 overrides `printSomething(A aParam)` from line 4.
   ☐ True ☐ False

2. `printSomething(B bParam)` on line 16 overrides `printSomething(A aParam)` from line 4.
   ☐ True ☐ False

3. Changing `B b = new B();` (line 24) to `A b = new B();` changes the output of `b.printSomething(b);` (line 33) (this is correct!). This happens as overloading is based on the static type of parameters.
   ☐ True ☐ False

4. Changing `B b = new B();` (line 24) to `A b = new B();` does not change the output of `b.printSomething(a);` (line 31) (this is correct!). This happens as Java uses Double Dispatch.
   ☐ True ☐ False

5. (Independent of the snippet above:) With dynamic dispatching, you may not be able to predict during compile time which method will be called (consider random runtime inputs).
   ☐ True ☐ False