Recapitulare

Obiective

Scopul acestui laborator îl constă recapitularea principalelor noțiuni de OOP și Java prezentate în cadrul laboratoarelor.

Testul final va verifica aceste concepte prin întrebări grilă similare celor date ca exemplu în acest laborator (nu toate exercițiile din laborator sunt însă conforme cu formatul testului).

Exerciții

1) Se consideră următoarea declarație de clasă. Ce se va întâmpla la compilarea și executarea ei?

```
public class Core
{
    private void privateMethod()
    {
        System.out.println("This is a private method");
    }

    public static void staticMethod()
    {
        System.out.println("This is a static method");

        this.privateMethod();
    }

    // Application Entry Point
    public static void main(String[] Params)
    {
        Core Instance = new Core();
        Instance.staticMethod();
        Core.staticMethod();
    }
}
```

- A. Se va afișa de 2 ori mesajul “`This is a static method`” și de două ori “`This is a private method`”
- B. Se va genera o eroare de compilare la ultima linie din `main()` (`Core.staticMethod()`)
- C. Se va genera o eroare de compilare la linia `this.privateMethod()`
- D. Se va arunca o excepție la executarea liniei `Instance.staticMethod()`

2) Se consideră următoarea situație. Câte versiuni distincte ale variabilei x sunt accesibile în (*)?

```
class Outer
```

```
{
    int x;

    class Inner extends Outer
    {
        int x;

        void f(int x)
        {
            (*)
        }
    }
}
```

- A. Una singură
- B. Doua
- C. Trei
- D. Patru

3) Se consideră următoarea structură de clase. La rularea metodei `printAll()`, ce valori se vor afișa și în ce ordine (ignorați faptul că acestea apar pe linii diferite)?

```
class Outer
{
    int x = 10;

    class Inner extends Outer
    {
        int x = 15;

        void printFirst(int x)
        {
            System.out.println(this.x);
            System.out.println(x);
            System.out.println(super.x);
        }

        void printSecond()
        {
            System.out.println(this.x);
            System.out.println(x);
            System.out.println(super.x);
        }

        void printAll()
        {
            printFirst(20);
            printSecond();
        }
    }
}
```

- A. 20, 15, 15, 10, 15, 20
- B. 10, 15, 15, 20, 20, 10
- C. 15, 20, 10, 15, 20, 10
- D. 15, 20, 10, 15, 15, 10

4) În următorul exemplu, ce se va afișa (ignorați faptul că mesajele apar pe linii diferite)?

```
public class Core
{
    private void someFunction(int x)
    {
        try
        {
            System.out.println("1");
            if (x == 1)
                return;
            x /= x;
        }
        catch (Exception error)
        {
            System.out.println("2");
        }
        finally
        {
            System.out.println("3");
        }
    }

    // Application Entry Point
    public static void main(String[] Params)
    {
       Core instance = new Core();
       instance.someFunction(1);
       instance.someFunction();
    }
}
```

- A. Nici un raspuns din cele de mai jos
- B. 1, 2, 3, 1, 3
- C. 1, 1, 3
- D. 1, 1, 2, 3

5) În fișierul “LuxuryCar.java” există următoarele declarații:

```
class Car {
   private String name = "Some Car";
}

public class LuxuryCar extends Car {
    public LuxuryCar() {
        name = "Luxury Car";
```

```
    }
}

class Main {
    // Application Entry Point
    public static void main(String[] params) {
       LuxuryCar instance = new Car();
    }
}
```

Ce se întâmplă în momentul în care încercăm să compilăm conținutul acestui fișier?

- A. Compilarea decurge fără probleme
- B. Compilarea eșuează cu 3 erori
- C. Compilarea eșuează cu 1 eroare
- D. Compilarea eșuează cu 2 erori

6) Fie următoarea secvență de cod. Ce se întâmplă la execuția ei?

```
import java.util.*;

public class Main
{
    private void process()
    {
        ArrayList<Integer> myList = new ArrayList<Integer>();

        try
        {
            for(int I = ; I < 5; ++I)
                myList.add(I + 1);

            int result = myList.get(5);
            System.out.println(result);
        }
        catch(StackOverflowError error)
        {
            System.out.println("! Stack overflow !");
        }
    }

    // Application Entry Point
    public static void main(String[] Params)
    {
       new Main().process();
    }
}
```

- A. Se va afișa “`! Stack overflow !`”
- B. Se va afișa 5
- C. Aplicația se va încheia normal (fără excepții), fără a afișa nici un mesaj

- D. Se va arunca o excepție de tipul `IndexOutOfBoundsException`, care nu va fi prinsă

7) Ce se întâmplă la compilarea și executarea următoarei aplicații Java?

```
public class Main
{
    private static Main Instance = new Main();

    public static Main getInstance()
    {
        return Instance;
    }

    public void print()
    {
        System.out.println(this.getClass());
    }

    // Application Entry Point
    public static void main(String[] Params)
    {
       Main.getInstance().print();
       new Main().print();
    }
}
```

- A. Nu apar erori sau excepții, nici la compilare nici la execuție
- B. Apare o eroare la compilare, deoarece nu se poate crea o instanță a clasei Main prin `new Main()`
- C. Apare o eroare la compilare, deoarece linia `new Main().print();` nu este corectă
- D. Apare o eroare la compilare, deoarece variabila `Instance` nu este accesibilă la nivelul funcției `getInstance()`

8) Ce va afișa instrucțiunea `new Child(2)`?

```
class Parent {
    public Parent() {
        System.out.println("Parent 0");
    }

    public Parent(int x) {
        System.out.println("Parent 1");
    }
}

class Child extends Parent {
    public Child(int x) {
        System.out.println("Child 1");
    }
}
```

9) Fie următoarea secvență:

```
class C {}

class D extends C {}

class A {
    void f(C c) {...}
    void f(D d) {...}
}

class B extends A {
    void f(C c) {...}
    void f(D d) {...}
}
```

Care implementare a metodei `f` va fi utilizată în cadrul invocării de mai jos:

```
A a = new B();
C c = new D();
a.f(c);
```

10) Adăugați un cuvânt cheie la următorul antet de clasă, astfel încât declarația să devină contradictorie:

```
 abstract class C
```

11) Întrebarea de mai sus, aplicată în cazul metodei:

```
abstract void f()
```

Resurse

- PDF laborator

Linkuri utile

1. Kathy Sierra, Bert Bates. *SCJP Sun Certified Programmer for Java™ 6 - Study Guide*. (available online) - o carte cu exerciții grilă și sumar pentru fiecare capitol, e.g. OOP Concepts, Inner Classes, Collections etc.
2. Subiecte și rezolvări teste grilă