DELFT UNIVERSITY OF TECHNOLOGY
Faculty of Electrical Engineering, Mathematics and Computer Science

TI1206 Object Oriented Programming (multiple choice part)
November 2nd, 2016, 9:00-11:00

Exam created by dr. A. Zaidman and checked by dr. G. Gousios

During this exam you are *not allowed* to make use of aids like your book or the slides. Digital devices such as phone, tablet, … stay in your bag and are turned off/silent.

Further information:
- Each question has 1 correct answer.
- There are 25 questions, each question has the same weight for determining your grade.
- This exam = 7 pages + 2 pages appendix.
- Fill in your answers on a separate form that will be processed automatically
    - Ask for a new form when you want to correct your answers on the form
    - Do not forget to fill in your name and student number on this form. Also, please sign it!
- The "version" and "day/evening" markers on the form should not be filled in.

Question 1 What is a "hit counter"?

a. An attribute of a class that we use to determine the number of spaces that are already "populated" in an array.
b. An attribute of a class that represents a set to keep track of how many times an identical element has been added to the set.
c. It is another name for a breakpoint.
d. **It is a way of controlling a breakpoint during debugging**.

Question 2 The cyclomatic complexity (CC) (which statement is **wrong**):

a. Is equal to the number of linearly independent paths through a piece of code, e.g. a method.
b. Is calculated by counting the number of conditions and adding 1 to it.
c. If a method has a cyclomatic complexity of *x*, there should be at most x tests testing that method.
d. Gives an indication that the method becomes hard to understand when the CC is 6 or higher.

Question 3 What happens in this piece of code? **[QUESTION SCRAPPED BECAUSE OF TYPO]**

```
public class IntList{

        private int[] element;
        private int number;

        public IntList(int n){
                int[] element = new int[n];
                number = 0;
        }

        public int getElement(int index)
        {
                if(index < number)
                        return element[i].
        }
}
```

a. This class will not compile. The Java compiler points to the constructor when giving an error.
b. This class will not compile. The Java compiler points tot he method getElement(int index) when giving an error.
c. This class compiles. A runtime exception occurs when executing the constructor.
d. This class compiles. A runtime exception occurs when executing the method getElement(int index)

Question 4 A design pattern is (which statement is **most complete and correct**)
a. A standard solution in terms of building blocks for programmers, a best practice that a developer can always use as is when confronted with a specific situation.
b. A standard solution in terms of building blocks for programmers, a best practice that a developer can sometimes use as is when confronted with a specific situation.
c. A standard solution in terms of building blocks for programmers, a best practice that a developer can use in specific situations, but that typically also requires tailoring to the situation.
d. A standard solution in terms of building blocks for programmers, a best practice that a developer can often use in a specific situation, but that typically also requires tailoring to the situation. Design patterns are know for their good design and good performance.

Question 5 Heisenbugs are:
a. Bugs that originate from the processor becoming too hot.
b. Bugs that originate from the fast cache memory in the processor being confused when doing multi-threading.
c. Bugs that are specific to processors that have the hyperthreading functionality, in that they chose the wrong registers just before or after switching threads.
d. Bugs that originate from wrong expectations w.r.t. the scheduling of threads.

Question 6 Method overloading means (which statement is the most correct one):
a. In 1 class you can define 2 or more methods with the same name.
b. In 1 class you can define 2 or more methods with the same name, but the names of the parameters need to be different.
c. In 1 class you can define 2 or more methods with the same name, but the types of the parameters need to be different.
d. In 1 class you can define 2 or more methods with the same name, but the parameter lists have to be different in terms of the number of parameters. If the number of paramters is

equal, than the types of the parameters need to be different or the order of the types needs to be different.

Question 7 Consider a class Point with 2 attributes x and y, both of type int. Now consider this equals method:

```
public boolean equals(Object other){
    if (other instanceof Point){
        Point that = (Point)other;
        return (this.x == that.x) && (this.y == that.y);
    }
    return false;
}
```

In a test method, the following 3 lines of code are written:

```
Point a = new Point(3,4);
Point b = new Point(3,4);
assertEquals(a, b);
```

Given that these snippets of code are placed in their right context, this code:

a. Will not compile.
b. Will lead to a runtime error.
c. Leads to a successful test. → "pass"
d. Leads to a failing test → "fail"

Question 8 Given the following code, what happens.

```
int[] row = {1,4,8};
int sum = 0;
for(int i = 0; i < row.length(); i = i + 1)
{
    sum = sum + row[i];
}
```

a. It will not compile.
b. It will lead to a runtime exception.
c. It will run perfectly. At the end of the execution, sum will have value 13.
d. It will run perfectly. At the end of the execution, sum will have the value 0.

Question 9 Given the following snippets of code:

```
// Person.java
public Person(String name)
{
    this.name = name;
    this.partner = new String("");
}

public static void marry(Person partner1, Person partner2)
{
    partner1.setMarriedTo(partner2);
    partner2.setMarriedTo(partner1);
}
```

```
// Program.java
public static void main(String[] args)
{
        Person p1 = new Person("Thomas");
        Person p2 = new Person("Linda");
        Person.marry(p1, p2);
        String partner = p1.getMarriedTo();
}
```

You can assume that all code compiles and works as expected. The question is what is the content of the string partner at the end of the main().

a. **The value of partner is "Linda". This is because of Java's pass by value approach where the address of p1 and p2 is copied into parameters partner1 and partner2 of the method marry(). Because Person is a reference type, p1 (p2) and partner1 (partner2) actually point to the same piece of memory and the content of that piece of memory can be manipulated through partner1 (partner2).**
b. The value of partner is "" (empty string). This is because of Java's pass by value approach where the address of p1 and p2 is copied into parameters partner1 and partner2 of marry(), but the changes made in the method marry are not "communicated back".
c. The value of partner is "Linda". This is because Java's copy back-and-forth principle that makes sure that changes made to parameters are always returned automatically.
d. None of the above.

Question 10 If a class A implements the interface Runnable, then:
a. Class A must implement a method *public static void main(String[] args)*
b. Class A must implement a method *public void start()*
c. **Class A must implement a method *public void run()***
d. Class A must have an attribute of type Thread.

Question 11 Given the following class and its instantiations:

```
public class Pair<A, B> {
        private A a;
        private B b;
}

Pair<String, Integer> p1 = new Pair<>("foo", 1);
Pair<Integer, String> p2 = new Pair<>(1, "foo");
```

What statement describes the situation best?
a. **System.out.println((p1 instanceof Pair)); //true**
   System.out.println((p2 instanceof Pair)); //true
   → because generic arguments are compile time only
b. System.out.println((p1 instanceof Pair)); //true
   System.out.println((p2 instanceof Pair)); //true
   → because generic arguments are runtime only
c. System.out.println((p1 instanceof Pair)); //false
   System.out.println((p2 instanceof Pair)); //false
   → because generic arguments are compile time only

d. System.out.println((p1 instanceof Pair)); //false
   System.out.println((p2 instanceof Pair)); //false
   → because generic arguments are runtime only

Question 12 Declaring a method **synchronized** indicates that

a. Only 1 thread can be active in that method of that instance of a class.
b. Only 1 thread can be active in that method or any other method of that instance of a class.
c. Only 1 thread can be active in that method or any other synchronized method of that instance of a class.
d. Only 1 thread can be active in any synchronized method of the program.

Question 13 With regard to events, one of the following statements is false.

a. An event always has a source and a listener.
b. Java ensures that the standard events have a listener interface and a corresponding adapter class (if there is more than 1 method in the listener interface).
c. A component can have 0, 1 or more listeners associated to it.
d. Events are important in the area of graphical user interfaces.

Question 14 Which statement is true? Making use of the keyword synchronized …

a. Makes sure that threads always work nicely together
b. Can lead to race conditions
c. Can lead to deadlock
d. Will always lead to significant speedups in the execution of your program.

Question 15 Java offers support to implement the Model View Controller pattern through:

a. The interfaces Model and View.
b. The classes Model and View.
c. The interfaces Model, View and Controller.
d. **The interface Observer and the class Observable**.

Question 16 Access control provides a mechanism to restrict who can access what in a class. Defining elements in a class as "protected" means:

a. That the class itself and its parent have access to these elements. To others, these elements are off limits.
b. The class itself, all entities in the package and its parent class have access to these elements. To others, these elements are off limits.
c. The class itself and all classes derived from it have access to these elements. To others, these elements are off limits.
d. The class itself, all classes derived from it, and classes in the same package have access to these elements. To others, these elements are off limits.

Question 17 What is the right construction order when making use of inheritance?

a. Call to the constructor of the super class
   Initialisation of the attributes
   Body of the constructor of the derived class
b. Initialisation of the attributes
   Call to the constructor of the derived class
   Body of the constructor of the derived class.
c. Body of the constructor of the derived class
   Constructor of the super class
   Initialisation of the attributes

d. Call to the constructor of the super class
   Body of the constructor of the derived class
   Initialisation of the attributes

Question 18 Consider the code in Appendix A. Given that code, what is the result of the following 3 lines of code?

```
Car car = new Hybrid("Toyota Prius", 5);
Hybrid prius = car;
System.out.println(prius.getBrand());
```

a. A compile-time error
b. A run-time error
c. No error, "Prius" is printed to the screen.
d. No error, "Toyota Prius" is printed to the screen.

Question 19 Consider the code in Appendix A. Given that code, what is the result of the following 3 lines of code?

```
Car car = new RangerExtender("Opel Ampera", 4);
ElectricCar leaf = (RangeExtender)car;
System.out.println(leaf.getBrand());
```

a. A compile-time error
b. A run-time error
c. No error, "Opel Ampera" is printed to the screen.
d. No error, "" is printed to the screen.

Question 20 Consider the code in Appendix A. Given that code, what is the result of the following 3 lines of code?

```
Car car = new ElectricCar("Renault Zoe", 4);
System.out.println(car.CO2_emission());
```

a. A compile-time error
b. A run-time error
c. No error, "Renault Zoe" is printed to the screen.
d. No error, "" is printed to the screen.

Question 21 Consider the following class, which statement is valid?

```
public class MultipleThreads implements Runnable {
        private Thread fThread;

        public MultipleThreads(){
                fThread = new Thread(this);
                fThread.start();
        }
        public void run(){
                System.out.println("Other thread starts");
        }
}
```

a. This piece of code will not compile.
b. Executing this piece of code will lead to a runtime error.
c. This piece of code will start a new thread and writes "Other thread starts" to the screen.
d. This piece of code will not start a new thread and writes "Other thread starts" to the screen.

Question 22 Which statement is **correct**?
a. You can declare a method finally, which indicates that it cannot be overridden.
b. You can declare a class finally, which indicates that no other classes can inherit from it.
c. You can declare a method final, which indicates that it cannot be overloaded.
d. You can declare a method final, which indicates that it cannot be overriden.

Question 23 The method "accept()" is (most correct/complete answer):

a. Part of the class Socket. It waits for an incoming request for communication.
b. Part of the class Socket. It waits for an incoming request for communication. Furthermore, this method "blocks" the program until there is an incoming request.
c. Part of the class ServerSocket. It waits for an incoming request for communication.
d. Part of the class ServerSocket. It waits for an incoming request for communication. Furthermore, this method "blocks" the thread until there is an incoming request.

Question 24 What happens for the next piece of code?

```
private static double fInterest = 3.15;

public static double giveInterest(double correctionfactor)
{
        return this.fInterest * correctionfactor;
}
```

a. **This will not compile**.
b. This will lead to a runtime error.
c. The return value is 6.30 if the parameter has the value 2.0
d. None of the above.

Question 25 When an exception is thrown (which statement is **correct**):
a. Execution in the surrounding try-block stops immediately (i.e., any statements after the statement raising the exception are skipped) and control resumes in the catch-block that matches the exception. After completing the catch-block, the finally block is executed (if present). Depending on the situation, control resumes after the catch block (or the finally block if present).
b. Execution in the surrounding try-block stops immediately (i.e., any statements after the statement raising the exception are skipped). Control resumes in the finally-block (if present), once the statements in the block have been executed, control goes to the catch-block that matches the exception. Depending on the situation, control resumes after the catch block (or the finally block if present).
c. Execution in the surrounding try-block stops immediately (i.e., any statements after the statement raising the exception are skipped) and control resumes in the catch-block that matches the exception. After completing the catch-block, the finally block is executed (if present). Depending on the situation, control resumes at the statement immediately after the statement in the try block raising the exception.
d. Execution in the surrounding try-block stops immediately (i.e., any statements after the statement raising the exception are skipped). Control resumes in the finally-block (if present), once the statements in the block have been executed, control goes to the catch-block that matches the exception. Depending on the situation, control resumes at the statement in the try block immediately after the statement raising the exception.

Appendix A

```
public abstract class Vehicle
{
	private String fBrand;
	private int fNumberOfDoors;

	public Vehicle(String brand, int numberOfDoors)
	{
		fBrand = brand;
		fNumberOfDoors = numberOfDoors;
	}

	public String getBrand()
	{
		return fBrand;
	}

	public int getNumberOfDoors()
	{
		return fNumberOfDoors;
	}
}

public interface OperatingRange
{
	public double getOperatingRange();
}

public class Car extends Vehicle
{
	public Car(String brand, int numberOfDoors)
	{
		super(brand, numberOfDoors);
	}
}

public class Hybrid extends Car implements OperatingRange
{
	private int fBatteryCapacity;

	public Hybrid(String brand, int numberOfDoors)
	{
		super(brand, numberOfDoors);
	}

	public double getOperatingRange()
	{
		return fBatteryCapacity / 4.5;
	}

}

public class RangeExtender extends Car
{
        public RangeExtender(String brand, int numberOfDoors)
        {
                super(brand, numberOfDoors);
        }
}
```

```
public class ElectricCar extends RangeExtender implements OperatingRange
{
       private int fBatteryCapacity;

       public ElectricCar(String brand, int numberOfDoors,
                      int batteryCapacity)
       {
              super(brand, numberOfDoors);
              fBatteryCapacity = batteryCapacity;
       }

       public String getBrand()
       {
              return “”;
       }

       public double getOperatingRange()
       {
              return fBatteryCapacity / 3.5;
       }

       public int CO2_emission()
       {
              return 0;
       }
}
```