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

TU Delft

CSE1100 Object Oriented Programming (multiple choice part)
November 1st, 2018, 9:00-11:00

Exam created by A. Zaidman and checked by F. Mulder

During this exam you are *not allowed* to make use of aids like your book or the slides. Digital devices such as phone, tablet, etc. stay in your bag and are turned off/silent. If you have a smartwatch, please also remove it and put it in your bag.

Further information:

- Each question has 1 correct answer.
- There are 25 questions. Each question has the same weight for determining your grade. No error correction will be applied.
- This exam consists of 9 pages (including the appendix)
- Fill in your answers on a separate form that will be processed automatically
  - Start by making the exam on a separate sheet, to avoid having to make corrections on the multiple-choice form.
  - Fill in the multiple-choice form in pencil (and not pen; a correction with pen might be misinterpreted when scanning your form!)
  - Do not forget to fill in your name and student number on this form.

Question 1 A *late propagation* is:

a. A technique that Java uses to copy the values of parameters only when they are actually used, thus being more efficient.
b. A consequence of Java's buffering of input/output, meaning that sometimes data is read/written late, i.e., after flushing a buffer.
c. A situation in which you copy/paste a piece of code, make changes to an instance, and only later make those changes to other instances of the cloning relation.
d. A late arrival of a message that is sent through a socket.

Question 2 In recent versions of Java (Java 8 and up) an interface can:

a. Not contain an implementation.
b. Contain static methods with an implementation.
c. Contain static final attributes
d. Contain static methods with an implementation and static final attributes.

Question 3 In recent versions of Java (Java 8 and up) an abstract class can:

a. Not contain an implementation.
b. Contain abstract methods with an implementation and non-abstract methods without an implementation.
c. Contain non-abstract methods with an implementation and abstract methods without an implementation.
d. Only contain abstract methods.

Question 4 Which statement is (most) correct?

a. A BufferedReader is a kind of Reader (extends Reader) and has a Reader as an attribute, but is typically more efficient because it reads bigger chunks into a buffer.
b. A BufferedReader is a kind of Reader (extends Reader) and has a Reader as an attribute, but is typically more efficient because it reads smaller chunks into a buffer.
c. A BufferedReader is a kind of Reader (extends Reader) and has a Reader as an attribute (in design pattern language this construction is called a "Decorator"), but it is typically more efficient because it reads smaller chunks into a buffer.
d. A BufferedReader is a kind of Reader (extends Reader) and has a Reader as an attribute (in design pattern language this construction is called a "Decorator"), but it is typically more efficient because it reads bigger chunks into a buffer.

Question 5 Which statement is true? The constructor without parameters:

a. Must always be declared public by the programmer
b. Must always be declared by the programmer
c. Can be declared private by the programmer
d. Cannot be tested

Question 6 Consider the following piece of code:

```
// in file Library.java
import java.util.ArrayList;

public class Library {
	private ArrayList<Book> books;
	private ArrayList<Person> borrowers;

	public Library() {
		ArrayList<Book> books = new ArrayList<Book>();
		ArrayList<Person> borrowers = new ArrayList<Person>();
	}

	public void addBook(Book b) {
		books.add(b);
	}

	public ArrayList<Book> retrieveBooksFromAuthor(String author) {
		ArrayList<Book> listOfBooks = new ArrayList<Book>();
		for(int i = 0; i < books.size(); i++) {
			if(books.get(i).getAuthor().equals(author))
				listOfBooks.add(books.get(i));
		}
		return listOfBooks;
	}
}


// in file LibraryTest.java
import org.junit.jupiter.api.Test;
import java.util.ArrayList;
import static org.junit.jupiter.api.Assertions.*;

public class LibraryTest {
	@Test
	public void testRetrieveBooks() {
		Library lib = new Library();
```

```
            Book b1 = new Book("X", "Y", "Z");
            Book b2 = new Book("A", "B", "C");
            lib.addBook(b1);
            lib.addBook(b2);
            ArrayList<Book> list = new ArrayList<Book>();
            list.add(b1);
            assertEquals(list, lib.retrieveBooksFromAuthor("X"));
      }
}
```

You can assume that the classes Book and Person are complete and all methods used in it are complete and work as expected. Knowing this, which statement is **true**?

a. The code that is shown will not compile.
b. The code compiles and the test will pass.
c. Executing the test "testRetrieveBooks()" will lead to a failing assert
d. Executing the test "testRetrieveBooks()" will lead to an exception

Question 7 Consider a class Person with 2 attributes name and age, respectively of type String and of type int.

```
public boolean equals(Object other){
    if (other instanceof Person){
        Person that = (Person)other;
        return ((this.name == that.name) && (this.age == that.age));
    }
    return false;
}
```

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

```
      Person a = new Person(new String("Tim"), 24);
      Person b = new Person(new String("Tim"), 24);
      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 Java uses *dynamic binding* :

a. To determine which implementation of a method that is present in an inheritance hierarchy to choose from at compile-time.
b. To determine which implementation of a method that is present in an inheritance hierarchy to choose from both at compile-time and run-time.
c. To determine which implementation of a method that is present in an inheritance hierarchy to choose from at run-time.

Question 9 Which statement is **false**? A *stack frame* contains:

a. Information w.r.t. to the object, i.e., an implicit reference to *this*
b. References to local variables and parameters for both primitive types and reference types.
c. References to local variables and parameters for reference types;
   actual data for local variables and parameters for primitive types

Question 10 Java's memory model makes use of a stack and a heap. Which statement about the stack is **true**?

a. A StackOverflowError is recoverable and can easily be solved at runtime by asking extra memory from the operating system.
b. A StackOverflowError can be caused by allocating a huge 2-dimensional array.
c. A StackOverflowError can be caused by a bad case of recursion.

Question 11 Consider the following code fragment. The methods hasNextLine() and getNextLine() are not defined to throw Exceptions (e.g. "throws IOException"). Which of the statements is true?

```
if(text != null & text.hasNextLine())
{
      System.out.println(text.getNextLine());
}
```

a. The compiler will not compile this piece of code.
b. The execution of this code will work without issues.
c. The execution of this code can lead to a StackOverflowError.
d. The execution of this code can lead to a NullPointerException.

Question 12 Consider the following code fragment. Which of the statements is true?

```
public static int[] copy(int[] row){
      int[] res = new int[row.length];
      for(int i = 0; i < row.length; i = i + 1)
            res[i] = row[i];
      return res;
}
```

a. This piece of code will not compile.
b. It will generate an ArrayOutOfBoundsException.
c. It will generate an ArrayIndexOutOfBoundsException.
d. It works.

Question 13 Consider the class 3DPoint and its swapXY method. What happens?

```
public class ThreeDPoint {
      private int x;
      private int y;
      private int z;

      public ThreeDPoint(int x, int y, int z) {
            this.x = x; this.y = y; this.z = z;
      }




      public static ThreeDPoint swapXY(ThreeDPoint p)
      {
            int temp = p.x;
            p.x = p.y;
            p.y = temp;
            return p;
```

```
        }

        public static void main(String[] args)
        {
              ThreeDPoint point = new ThreeDPoint(1, 2, 3);
              System.out.println(point.x + ", " + point.y + ", " +
                  point.z);
              point = ThreeDPoint.swapXY(point);
              System.out.println(point.x + ", " + point.y + ", " +
                  point.z);
        }
}
```

a. This will not compile because x and y are private and cannot be accessed from within a static method.
b. After execution 1, 2, 3 and 1, 2, 3 will be on the screen.
c. After execution 1, 2, 3 and 2, 1, 3 will be on the screen.
d. None of the above

Question 14 Which statement is **false?** The finally clause of a try/catch block:

a. Is optional.
b. Is typically a good idea if you want to clean up resources like network connections or file readers.
c. Will be executed after an exception is thrown in the try block, but before the catch block is executed.
d. Will be executed after an exception is thrown in the try block, after the catch block has been executed.

Question 15 When it comes to Java generics, we can say:

a. Generic arguments are compile-time only, and no constraint can be put on generic type arguments.
b. Generic arguments are compile-time only, and constraints can be put on generic type arguments.
c. Generic arguments are both compile-time and execution-time, and no constraint can be put on generic type arguments.
d. Generic arguments are both compile-time and execution-time, and constraints can be put on generic type arguments.

Question 16 A Heisenbug is:

a. A specific type of "bug" that can appear due to expectations with regard to the scheduling of processes by the operating system.
b. A specific type of "bug" that can appear due to expectations with regard to the scheduling of processes by the Java Virtual Machine.
c. A specific type of "bug" that can appear due to expectations with regard to the scheduling of threads by the operating system.
d. A specific type of "bug" that can appear due to expectations with regard to the scheduling of threads by the Java Virtual Machine.

Question 17 Consider the following statements about the method accept() of the class ServerSocket. Which statement is correct and complete?

a. It accepts incoming connections from Socket classes.
b. It accepts incoming connects from Socket classes. It is blocking, meaning that the program waits until a connection is actually made.
c. It accepts incoming connections from ServerSocket classes.
d. It accepts incoming connects from ServerSocket classes. It is blocking, meaning that the program waits until a connection is actually made.

Question 18 In case your class already inherits from some class, you can still work with a separate thread by:

a. Having your class extend from the abstract Runnable class and implementing all the necessary methods. Subsequently, you need to create a new Thread object, to which you pass an instance of your class. Finally, you start the thread by calling the start() method of the new Thread object.
b. Having your class implement the Runnable interface and implementing all the necessary methods. Subsequently, you need to create a new Thread object, to which you pass an instance of your class. Finally, you start the thread by calling the start() method of the new Thread object.
c. Having your class extend from the Runnable class and implementing all the necessary methods. Subsequently, you need to create a new Thread object, to which you pass an instance of your class. Finally, you start the thread by calling the run() method of the new Thread object.
d. Having your class implement the Runnable interface and implementing all the necessary methods. Subsequently, you need to create a new Thread object, to which you pass an instance of your class. Finally, you start the thread by calling the run() method of the new Thread object.

Question 19 If you declare a method *synchronized* this means that:

a. There can be only 1 thread active in all objects of that class.
b. There can be only 1 thread active in one of the synchronized methods of all objects of that class.
c. There can be only 1 thread active in a particular object of that class.
d. There can be only 1 thread active in synchronized methods of a particular object of that class.

Question 20 The keyword "default" is used to:

a. Indicate a case in a switch statement that matches when none of the other options are chosen.
b. Indicate that an exception should not be caught, but should be send straight to the default exception handler.
c. Indicate a default constructor
d. Indicate that attributes should have a default value, if no explicit call to a constructor is made.

Question 21 Java uses the "pass by value" principle. Which statement is **false**?

a. When passing values of primitive types as parameters to a method, these values are copied into the stackframe belonging to the called method.
b. When passing objects as parameters to a method, the memory addresses of these objects are copied into the stackframe belonging to the called method. The actual objects reside in the heap memory and are read-only.
c. When passing objects as parameters to a method, the memory addresses of these objects are copied into the stackframe belonging to the called method. The actual objects reside in the heap memory.
d. When passing an array as parameter to a method, the memory address of the array is copied into the stackframe belonging to the called method. The actual values stored in the array reside in the heap memory.

Question 22 Consider the following snippets of code

```
public class Person {
  private String name;
  private ArrayList<Car> cars;
}

public class Car {
  private String licenseplate;
  private Person owner;
}
```

The relationship between these two classes can be called:

a. 1 to 1 relation, reflexive
b. 1 to many relation, reflexive
c. 1 to 1 relation, non-reflexive
d. 1 to many relation, non-reflexive

Question 23 Consider the code in Appendix A. You can assume that the code in Appendix A compiles. Given that code, what is the result of the following lines of code?

```
Tablet ipad = new TabletWithPencil("iPad Pro", "A10", 64, 10.5, false);
System.out.println(ipad.getVersion());
```

a. A compile-time error
b. A run-time error
c. No error, “5.0” is printed on the screen
d. None of the above

Question 24 Consider the code in Appendix A. You can assume that the code in Appendix A compiles. Given that code, what is the result of the following lines of code?

```
MobileComputerDevice mobile = new Smartphone("iPhone 8", "A11", 64,
    4.7, "4G", true);
MobileDevice device = mobile;
System.out.println(device.hasFingerprintsensor());
```

a. A compile-time error, indicating the 2nd line
b. A compile-time error, indicating the 3rd line
c. No error, “true” is printed on the screen
d. No error, “false” is printed on the screen

Question 25 Consider the code in Appendix A. You can assume that the code in Appendix A compiles. Given that code, what is the result of the following lines of code?

```
MobileDevice watch = new Smartwatch("Apple Watch Series 4", "W4", 8, true);
Tablet device = (Tablet)watch;
System.out.println(((Smartwatch)watch).hasGPS());
```

a. A compile-time error, indicating the 2nd line
b. A compile-time error, indicating the 3rd line
c. A run-time error
d. No error, “true” is printed on the screen

Appendix A

```
public interface Bluetooth {

	public String getVersion();
}

public abstract class MobileDevice {

	private String processor;
	private int memory;
	private String model;

	public MobileDevice(String model, String processor, int memory) {
		this.model = model;
		this.processor = processor;
		this.memory = memory;
	}
}

public class MobileComputerDevice extends MobileDevice {

	private double screensize;

	public MobileComputerDevice(String model, String processor, int memory,
	    double screensize) {
		super(model, processor, memory);
		this.screensize = screensize;
	}

	public boolean hasFingerprintsensor() {
		return false;
	}
}

public class Smartphone extends MobileComputerDevice {

	private String network;
	private boolean fingerprintsensor;

	public Smartphone(String model, String processor, int memory,
	    double screensize, String network, boolean fingerprintsensor) {
		super(model, processor, memory, screensize);
		this.network = network;
		this.fingerprintsensor = fingerprintsensor;
	}


	public boolean hasFingerprintsensor() {
		return this.fingerprintsensor;
	}
}
```

```
public class Smartwatch extends MobileDevice {

	private boolean gps;

	public Smartwatch(String model, String processor, int memory,
	    boolean gps) {
	       super(model, processor, memory);
	       this.gps = gps;
	}

	public boolean hasGPS() {
	       return this.gps;
	}
}

public class Tablet extends MobileComputerDevice {

	private boolean mobilenetwork;

	public Tablet(String model, String processor, int memory,
	    double screensize, boolean mobilenetwork) {
	       super(model, processor, memory, screensize);
	       this.mobilenetwork = mobilenetwork;
	}

	public boolean hasMobilenetwork() {
	       return this.mobilenetwork;
	}
}

public class TabletWithPencil extends Tablet implements Bluetooth {

	public TabletWithPencil(String model, String processor, int memory,
	    double screensize, boolean mobilenetwork){
	       super(model, processor, memory, screensize, mobilenetwork);
	}

	public String getVersion(){
	       return "5.0";
	}
}
```