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

TU Delft

CSE1100 Object Oriented Programming (multiple choice part)

November 8th, 2019, 09:00-11:00

Exam created by A. Zaidman and T. Overklift

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 has 10 pages (including 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 the form.

1. Which statement about Object-oriented programming (OOP) is **false**?
    A. When using OOP a program typically consists of classes, objects and statements.
    B. OOP typically uses immutable data (unchangeable objects).
    C. When using OOP, several methods and relevant data are typically encapsulated in a class.
    D. Inheritance is a frequently used concept within OOP.

2. If the cyclomatic complexity of a method is 3 this means:
    A. There are 2 linearly independent paths through this method.
    B. There should be at most 3 tests testing the method.
    C. This indicates that the method is hard to understand.
    D. None of the above.

3. Which of the following is an outcome of defining an implemented method body in an abstract class?
    A. Child classes must convert this method to an abstract method.
    B. Concrete child classes must override this method.
    C. Child classes can no longer redefine the body of this method.
    D. Child classes no longer need to define this method.

4. 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.age == that.age)
                     && (this.name == that.name));
         }
         return false;
   }
   ```

   In a test method, the following 4 lines of code are written:

   ```
   String name = new String("Tim");
   Person a = new Person(name, 24);
   Person b = new Person(name, 24);
   assertEquals(a, b);
   ```

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

   A. Will not compile.
   B. Leads to a successful test. → "pass"
   C. Leads to a failing test → "fail"
   D. Leads to a runtime exception during the test → "error"

5. Is it considered good practice to set the visibility of class attributes to `public` in Java?
   A. Yes, because this way the children of the class can easily access the properties of their parent.
   B. Yes, because this way external classes can more easily change the values of these attributes.
   C. No, because it is best to set the visibility of all class attributes to `protected`.
   D. No, because this way other classes can modify these attributes without any limitation or checks.

6. Which of the following statements regarding the use of generics and parameterized types in Java is **false**?
   A. Generics and parameterized types make source code more type safe by checking for this during run time.
   B. When using generics, the types that can be used as a parameterized type can be limited with type constraints.
   C. Generics and parameterized types reduce the need for casts when retrieving items from a structure such as an `ArrayList`.
   D. Generics and parameterized types are not compatible with primitive types, but Java deals with this through the concept of "autoboxing".

7. Consider the following lines of code:

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

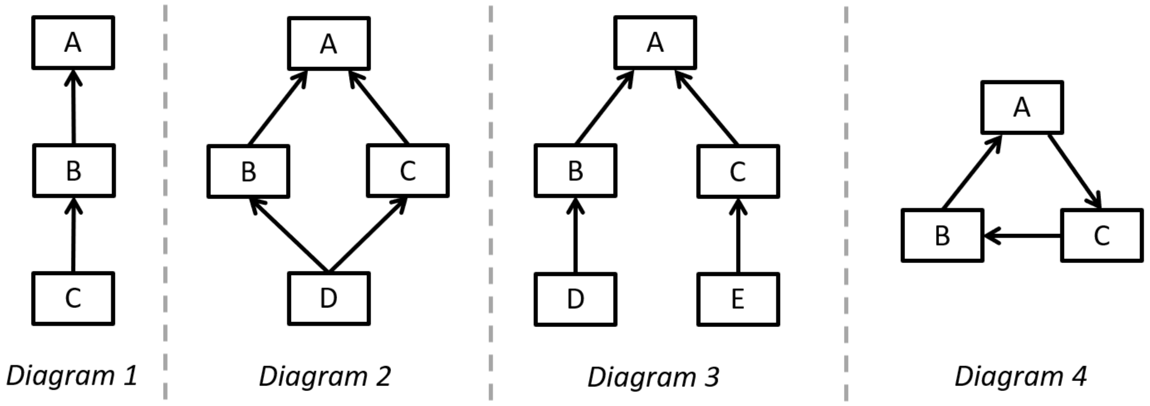
Given that this snippet of code is placed in a correct context and that we have the following lines in a main method:

```
int[] ar1 = new int[10];
int[] ar2 = copy(ar1);
int[] ar3 = ar1;
```

We can derive that

A. ar1, ar2 & ar3 all point to the same memory location.
B. ar1 & ar2 point to the same memory location but ar3 points to a different memory location.
C. ar1 & ar3 point to the same memory location but ar2 points to a different memory location.
D. ar1, ar2 & ar3 all point to different memory locations.

8. Study the figure with inheritance relations between classes below. An arrow in this diagram indicates an inheritance relation (e.g. in diagram 1: B is a child of A).



Which of the diagrams in the figure best represents a situation that can only occur when allowing *multiple inheritance*?

A. Diagram 1
B. Diagram 2
C. Diagram 3
D. Diagram 4

9. Consider the following block of code:

```
try(Scanner sc = new Scanner(System.in)){
     String inString = sc.next();
     int i = Integer.parseInt(inString);
     System.out.print("You typed the number: " + i);
}
catch (NumberFormatException exception){
     System.out.println("Not a number!");
}
```

Given that this snippet of code is placed in a correct context, what is the expected result when this code snippet is executed, and the user types "three"?

A. The program prints: "Not a number!". All resources are closed properly.
B. The program prints: "You typed the number: 3". All resources are closed properly.
C. The program prints: "Not a number!". The scanner is not closed properly afterwards because of the exception that was raised.
D. The program prints: "You typed the number: Not a number!". All resources are closed properly.

10. Which of the following statements about error and exception handling is **false**?

A. Many nested recursive calls may lead to an `StackOverflowError`.
B. Errors are typically more severe than Exceptions, and are hard to recover from during run time.
C. You can use a try block on its own, without a resource, finally clause, or catch clause(s).
D. Java will not compile when a checked exception is neither handled in a catch clause nor added to the method signature with the `throws` keyword.

11. Which statement about `BufferedReader` and `Scanner` is **false**?

A. A `Scanner` can be used to read from files as well as to scan strings.
B. A `BufferedReader` is a Reader that takes a Reader as an argument.
C. A `Scanner` has built in support for all primitive types.
D. A `BufferedReader` only has methods that return a single character at a time or return an array of characters with a predefined length.

12. Which statement about `abstract` classes or methods is **false**?

A. An `abstract` method should be in an abstract class or interface.
B. An `abstract` class can only contain abstract methods.
C. An `abstract` method has no method body.
D. An `abstract` class can contain attributes.

13. Which statement about `Optional` is **false**?

A. An `Optional` is a container that can either contain another `Object` or be *empty*.
B. Using `Optional` is the only way to avoid a `NullPointerException` being raised other than catching it.
C. Using `Optional` tends to make code more concise, and reduces the need for explicit `null` checks
D. An `Optional` can be used in combination with a default return value by using `orElse()`.

14. 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**.

15. Consider the class ThreadWorker and a main() method below.

```
public class ThreadWorker implements Runnable {

     public ThreadWorker() { }

     public void run() {
          System.out.println("A new thread writes this");
     }
}
```

The main method:

```
public static void main(String[] args) {
     ThreadWorker tw = new ThreadWorker();
     tw.run();
}
```

Which statement is valid?

A. The class `ThreadWorker` does not compile.
B. The compiler gives an error when trying to compile the `main()` method.
C. The code executes as expected and “A new thread writes this” is indeed printed by a newly started 2nd thread.
D. **None of the above**.

16. Which of the following statements regarding the `static` keyword is **false**?
    A. `static` methods can only refer to other methods and variables in a `static` context.
    B. `static` methods can never refer to `this`.
    C. When multiple objects of a class are instantiated, each object gets its own copy of any `static` class attributes.
    D. `static` methods can be overloaded but `static` methods cannot be overridden.

17. The concept of deadlock can best be described as:
    A. Two processes are fighting for the same resource, e.g., a file, but once one of the processes gains a lock on the resource, the other process gets locked out.
    B. When, in a multiple inheritance situation, a class inherits from two other classes that both have an attribute of the same name and type, the compiler does not know which attribute to inherit resulting in a lock out.
    C. **Two threads that are waiting for each other because both of them are working in `synchronized` sections of code, but they are both waiting for the other thread to release a `synchronized` lock**.
    D. None of the above.

18. Consider the access control rules. What observation about `package` access is correct?
    A. When using the `package` modifier you give access to the class itself and all other classes declared in the same package.
    B. When using the `package` modifier you give access to the class itself, its children, and all other classes in the package.
    C. When using no modifier you give access to the class itself and all other classes declared in the same package.
    D. When using no modifier you give access to the class itself, its children, and all other classes declared in the same package.

19. Which statement about testing constructors is **correct**?
    A. The constructor should not be tested.
    B. The constructor can only be sufficiently tested if there is an equals method of the class.
    C. The constructor can only be sufficiently tested if there are getter methods defined for each attribute of the class.
    D. None of the above.

20. Which statement is **false**? Checking whether a method throws an exception during testing:
    A. Is never required to reach 100% test coverage for that method.
    B. Can usually be done with the `assertThrows()` assertion that checks whether a specific exception is thrown.
    C. Cannot be done with the `assertThrows()` assertion when an exception is immediately caught.
    D. Is an example of “bad weather testing”.

21. Consider a `main` method and the method `swapLocal` below.

```
public static void main(String[] args) {
     int[] row = {1,2,3,4,5,6,7,8,9,10};
     swapLocal(row);
     for(int i = 0; i < row.length; i++) {
          System.out.print(row[i] + ", ");
     }
}

public static int[] swapLocal(int[] row) {
     int[] returnrow = new int[2*row.length];
     if((row.length % 2 == 0) && (row.length != 0)) {
          for(int i = 0; i < row.length; i = i + 2) {
               int temp = row[i];
               row[i] = row[i+1];
               row[i+1] = temp;
          }
     }
     return returnrow;
}
```

What will the result of executing this `main` method be?

A. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
B. 2, 1, 4, 3, 6, 5, 8, 7, 10, 9,
C. 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
D. None of the above

22. Which statement about streams (e.g. `IntStream.range(0, 4)`)in Java is **false**?

A. You can define an infinite stream.
B. Streams can only be traversed once.
C. Performing a filtering operation on a stream changes the input.
D. You can execute streams in serial or in parallel.

23. Consider the code in *appendix A*. When the main method is executed, what is the expected output of the program?

    A. `A pair of sharks`
       `A pair of ducks`
       `A pair of animals`

    B. **`A pair of sharks`**
       `A pair of animals`
       `A pair of animals`

    C. `A pair of sharks`
       `A pair of ducks`
       `A pair of sharks`

    D. `A pair of sharks`
       `A pair of animals`
       `A pair of sharks`

24. Consider the code in *appendix B*. When running the `replaceNonexistingPersonTest()` method, the code:

    A. Will not compile.
    B. Leads to a successful test. → "pass"
    C. Leads to a failing test → "fail"
    D. Leads to a runtime exception during the test → "error"

25. Consider the code in *appendix B*. With the information in the appendix we can say the following about the class `Apartment`:

    A. `Apartment` is a child class of some parent, and `Apartment` is a parent class of some child.
    B. `Apartment` is a child class of some parent, but we can't say anything about possible children of `Apartment`.
    C. `Apartment` is a parent class of some child, but we can't say anything about possible parents of `Apartment`.
    D. We can't say anything about possible parents or children of `Apartment`.

```java
public class Animal {

	public void makePair(Animal a) {
		System.out.println("A pair of animals");
	}
}

public class Shark extends Animal{

	public void makePair(Shark s) {
		System.out.println("A pair of sharks");
	}
}

public class Duck extends Animal {

	public void makePair(Duck d) {
		System.out.println("A pair of ducks");
	}
}

public class Main {

	public static void main(String[] args) {
		Shark bruce = new Shark();
		Shark anchor = new Shark();
		Animal donald = new Duck();
		Duck daffy = new Duck();

		bruce.makePair(anchor);
		donald.makePair(daffy);
		anchor.makePair(daffy);
	}
}
```

Appendix A

```java
class Apartment {
    private String location;
    private String[] people;
    private int amount;

    public Apartment(String location, int nPeople) {
        if (nPeople < 0) {
            nPeople = 0;
        }
        this.location = location;
        people = new String[nPeople];
        amount = 0;
    }

    public String[] getPeople() {
        return this.people;
    }

    public void addPerson(String name) {
        if(amount < people.length) {
            people[amount] = name;
            amount++;
        } else {
            throw new ArrayIndexOutOfBoundsException("Apartment is full!");
        }
    }

    public void replacePerson(String replace, String with) {
        for (int i = 0; i < people.length; i++) {
            if(people[i].equals(replace)) {
                people[i] = with;
            }
        }
    }
}
```

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class ApartmentTest {

    @Test
    public void replaceNonexistingPersonTest() {
        Apartment o = new Apartment("Stieltjesweg 546", 4);
        o.addPerson("Stefan");
        o.addPerson("Sander");
        o.replacePerson("Frank", "Otto");
        String[] people = o.getPeople();
        assertEquals("Stefan", people[0]);
    }
}
```

Appendix B