voorbeeldexamen OGP - ingevuld

autheur: JVM

Wat ingevuld moet worden staat in **vet** (aka, kon ik zo goed als letterlijk in cursus vinden).
Waarbij ik twijfel staat in ***cursief vet*** (aka, kon ik uit de cursus afleiden, maar ben ik niet zeker van).

Question 1: Fill in the blanks.

A decomposition of a software system is modular if each module can be **developed**, **understood**, **verified**, and **evolved** independently from and in parallel with the other modules.
The main approach for managing the complexity of developing software systems is by achieving modularity through **decomposition**.
This means the system is split into a **client** module that implements the system's functionality in a programming language extended with additional **operations** (this approach is called **procedural abstraction**) or additional **data types** (this approach is called **data abstraction**),
and a module that implements the **abstractions** using the constructs of the base programming language.
This approach works best if the **abstraction** is documented sufficiently **precisely** and **abstractly**.

Instance fields can further be categorized as **immutable** if an object's **abstract value** is fixed at construction time, or **mutable** if it can change during the object's lifetime.

Formal class documentation includes public **class invariants** which define the valid **abstract value** of an instance and private **class invariants** which define the valid **abstract state** of an instance.

Constructors and methods are documented mainly using **javadoc** which specify results and side-effects, and **preconditions** (in case of contractual programming) or **throws clauses** (in case of defensive programming).

Multi-object abstractions typically involve **bidirectional** associations, whose **consistency** must be preserved at all times.
This means that if according to an object o1's representation, o1 is associated with an object o2, then ***o2's representation includes it is associated with object o1***.

Inheritance means that classes can be declared as **subclasses** of other classes. In that case, instances of the **subclass** are also considered to be instances of the **superclass**. A class that only serves as a generalization of other classes and that is not intended to be instantiated directly is called a/an **abstract** class. A **polymorphic** variable is one that can refer to objects of different **subclasses**.

Java’s **type checker** allows a field access or a method call only if the target expression’s **type** declares or inherits the field or method.
Programmers can work around this by using **typecasts**. These check an object’s class at **run time**; if the check fails, it is reported as a/an **ClassCastException**.

If a class declares a method whose name and number of types of parameters match a method of the **superclass**, then this method is said to **override** the other one.
A method that only serves to be **overriden** can be declared **abstract**; for such methods, you do not need to provide a/an **implementation**.

There are two kinds of method binding: in case of **dynamic** binding, the method to be executed is determined at **run time** based on the **dynamic type** of the target expression of the call; in case of **static** binding, the method to be executed is determined at **compile time** based on the **static type** of the target expression of the call.