Midterm Review

Wednesday, October 11, 2023 9:06 AM

Exam is on 10/23/23

Terms covered (So far):

4 Pillars of OOD
- Inheritance
  - Base class, subclasses inherit it's methods and variables.
- Abstraction
  - Prevents a class from being instantiated, or make it so any subclasses must implement a method that was abstract in the class it inherits from.
- Polymorphism
  - Allows for references to multiple types of objects that inherit from a specific type.
- Enapsulation
  - Allocation of responsibility between the user and programmer, and protection of data.

The 4 C's of an interface
- Complete
  - Class must implement all the methods, even if it doesn't use them
- Convenience
  - Class must be easy enough to use for the user, or other programmers using that class
- Consistent
  - Class must have coding style/technique/choices that stay the same throughout the class.
- Cohesive
  - Class must have understandable names for it's variables and classes.

Overloading vs. Overriding
- Overloading is two methods with the same name but different signatures within the same scope
- Overriding is two methods with the same name and signature within an inheritance structure.

The patterns
- Use/a
  - Association: Temporarily associates with another object.
- Is/a
  - Inheritance
- Has/a - Aggregation
- Has/a - Composition
  - A member variable from this class is an object of another class

Static vs. Dynamic type

Static vs. Dynamic binding

Class invariants
- A choice made by the class that all it's methods must follow.

Coupling

The singleton pattern
- Only one instance of an object is allowed to be created.

Aggregation, composition

Hiding implementation details from the user
And the complexity

Has/a relationship between two objects where it is the weaker one.

If one object is destroyed, it does not destroy the other.

In composition, it is the opposite. It is the stronger one in the relationship, but it is also destroyed if the other is.

Instanceof must always be used before type-casting because it could lead to a runtime error. This is because you might try to type-cast into a class that isn't the right type.

Encapsulation: Prevention of data access to the user and allocation of responsibility between the program and user.
Abstraction: Hiding of implementation and complexity detailings from the user.
Polymorphism: Accessing different classes through a general classes that has all of the subclasses being accessed defined by it.
Inheritance: Defining subclasses based off a base-class, allowing the subclass to automatically define the base-classes methods and variables.

Static binding: Binding a method call to a method using the compiler before the start of the program.
Dynamic binding: Binding a method call to a method during the runtime of the program.

Static type: The type of the variable that does change.
Daycare numberVar = new Daycare();

Dynamic type: The object type that the variable refers to.
Daycare numberVar = new Daycare();

Singleton:
Only allows a single object of a class to be instantiated in memory.
Private static ClassName uniqueInstance;
Private constructors
Public static ClassName uniqueInstance() {
If (uniqueInstance == null) {
    uniqueInstance = new ClassName();
}
Return uniqueInstance;
}

Continuous
- Use the same style throughout the program

Convenient
- Make the program easy to use and understand

Complete
- Define all variables in methods even if not used

Cohesive
- Use good variable names

Interface: Only defines the methods that a class that uses that interface must define. Abstract still has to be shown in memory even if it cant be instantiated, because it can still define member variables that can be inherited from.

Instanceof must always be checked before a type-cast! This prevents a runtime error.

Use-a: Association
- Temporarily creates an object that holds data to use it, then discards immediately after.

Is-a: Inheritance
- Inherits the base class's data to use in the sub-class.

Has-a: Composition
- Stronger relationship, the smaller object is a member variable in the larger object. Strongly coupled.

Has-a: Aggregation
- Weaker relationship, the smaller object has variables that are defined using the data within the larger object's class. Weakly coupled.

Overriding - Two methods with the same name and signature, but defined within an inheritance structure so that the sub-class method replaces the base method.
Overloading- Two methods with the same name and scope, but have different signatures.

Polymorphic reference: When a variable is instantiated using an object that is higher in the inheritance chain to another class.