Christopher Hernandez
Week 5 Research

1. What are the four pillars of Object-Oriented Programming? Explain each pillar.

(1) Encapsulation: Encapsulation refers to the practice of hiding the implementation details of an object from the outside world and exposing only what is necessary for the object's proper functioning. In other words, encapsulation is the practice of bundling data and methods that operate on that data within a single unit or object. The idea behind encapsulation is to prevent external code from directly manipulating an object's data and ensure that the object's state is always valid and consistent.

(2) Abstraction: Abstraction refers to the practice of hiding unnecessary details and complexity while emphasizing essential features. Abstraction enables programmers to think at a higher level of conceptualization and work with objects in terms of their essential characteristics rather than their implementation details. Abstraction is achieved through the use of abstract classes, interfaces, and other programming constructs.

(3) Inheritance: Inheritance allows programmers to create new classes that are a modified version of existing classes. Inheritance is based on the idea of the is-a relationship, where a new class is a specialized version of an existing class. Inheritance enables code reuse, promotes code organization, and allows programmers to model complex relationships between objects.

(4) Polymorphism: Polymorphism refers to the ability of objects of different types to be used interchangeably. Polymorphism enables programmers to write generic code that can work with objects of different types without knowing their specific type at compile time. Polymorphism is achieved through the use of interfaces, abstract classes, and method overloading/overriding.

2. What is the relationship between a Class and an Object?

- a class is a blueprint or template for creating objects, while an object is an instance of a class. A class defines a set of attributes and behaviors that are shared by all instances of that class, while an object is a specific instance of a class that has its own unique values for those attributes.

3. What are the differences between checked and unchecked exceptions?

(1) Checked Exceptions: Must be declared in a method or handled in a try-catch block. These exceptions are checked at compile-time, and the compiler ensures that they are either declared or handled. This is done to enforce error handling and ensure that exceptions are not ignored by the programmer. Examples of checked exceptions include IOException, SQLException, and ClassNotFoundException.

(2) Unchecked Exceptions: Are not required to be declared or handled. These exceptions are not checked at compile-time, and the programmer is not required to handle them explicitly.

Examples of unchecked exceptions include NullPointerException, ArrayIndexOutOfBoundsException, and ArithmeticException.

4. What are the differences between abstract classes and interfaces? When should you use one over the other?

Abstract	Interfaces
An abstract class is a class that cannot be instantiated, and it is meant to be extended by other classes.	An interface is a collection of abstract methods and constants, which defines a contract for a class to implement.
Abstract classes can contain both abstract and concrete methods, as well as instance variables.	Interfaces cannot contain instance variables, and all methods declared in an interface are by default public and abstract.
Abstract classes can also provide default implementations for some or all of their methods.	A class can implement multiple interfaces, but it cannot extend multiple classes.
A class can extend only one abstract class, but it can implement multiple interfaces.	Interfaces are used when you want to define a common set of methods or behavior that can be implemented by different classes, without worrying about their specific implementation details.
Abstract classes are used when you want to create a base class that provides a common set of methods or behavior that can be shared by multiple subclasses.	Interfaces are also useful for achieving polymorphism, as they allow objects of different classes to be treated as if they share a common interface.
Use when:	Use when:
If you want to provide a default implementation of some methods and/or instance variables, use an abstract class.	If you want to define a contract for behavior and/or ensure that multiple classes share a common interface, use an interface
If you need to create a base class for other classes to inherit from, use an abstract class.	If you need to support multiple inheritance, use interfaces.

5. What is unit testing and why is it important?

The purpose of unit testing is to verify that each unit of the application is working correctly and producing the expected results. Unit testing is important for several reasons:

(1)Early detection of defects: Unit testing can help detect defects early in the development process, before they become more difficult and costly to fix. By testing individual units in isolation, defects can be identified and fixed quickly and efficiently.

(2)Faster feedback: Unit testing provides quick feedback on the correctness of individual units, which helps developers identify and fix defects more quickly.

(3)Improved code quality: Unit testing helps improve code quality by encouraging developers to write testable code and to consider edge cases and boundary conditions.

(4)Facilitates refactoring: Unit testing makes it easier to refactor code by ensuring that existing functionality is not broken when changes are made.

(5)Documentation: Unit tests can serve as a form of documentation for the code, helping other developers understand how the code is intended to work.

(6)

6. What is your favorite thing you learned this week? I only have fun doing this type of work now since I cant grasp the concepts any longer with Java. This week was way too loaded and I feel we stuffed 3 weeks into one 0.0

References:

https://docs.oracle.com/javase/tutorial/java/concepts/interface.html

https://docs.oracle.com/javase/tutorial/java/concepts/index.html

https://docs.oracle.com/javase/tutorial/java/concepts/class.html

https://docs.oracle.com/javase/tutorial/java/concepts/object.html

https://docs.oracle.com/javase/tutorial/java/concepts/interface.html