

Grade 12 Object-Oriented Programming Study Guide

This study guide provides a comprehensive review of Object-Oriented Programming (OOP) concepts, specifically tailored for Grade 12 Information Technology. It synthesizes theoretical principles, practical implementation details, and common assessment criteria found in the source context.

Part 1: Review Quiz

Instructions: Answer the following questions in 2-3 sentences based on the principles of Object-Oriented Programming.

1. Define a "Class" in the context of OOP and explain its relationship to an "Object."
2. Describe the primary difference between an "Accessor" method and a "Mutator" method.
3. What is the purpose of the `super()` method call within a child class constructor?
4. Explain the difference between "Static" and "Non-Static" methods regarding how they are invoked.
5. How does "Method Overloading" differ from "Method Overriding"?
6. Distinguish between "Information Hiding" and "Encapsulation."
7. Identify two major advantages of using "Inheritance" in a software project.
8. What is a "Constant Field," and how is it typically declared in Java?
9. Explain the "has-a" relationship associated with "Composition" versus the "is-a" relationship of "Inheritance."
10. What are "Parallel Arrays," and why are they generally considered an inferior data structure compared to an array of objects?

Part 2: Answer Key

1. **A Class** is a user-defined blueprint or template from which objects are created. **An Object** is a specific instance of that class, possessing its own state (attributes) and behavior (methods).
2. **An Accessor method** (getter) is a typed method used to return the value of a private field to a calling program. **A Mutator method** (setter) is a void method used to change or modify the value of a private field in a controlled manner.

3. **The super() call** is used to invoke the constructor of the parent or superclass. It passes parameters to the parent class to initialize its private fields before the child class performs its own initialization.
4. **Static methods** are bound to the class itself and are called using the class name without needing to instantiate an object. **Non-static methods** are bound to specific instances and require an object of the class to be created before they can be invoked.
5. **Method Overloading** occurs when two methods in the same class have the same name but different parameter lists. **Method Overriding** occurs when a child class provides a specific implementation for a method that already exists in its parent class with the same name and signature.
6. **Encapsulation** is the broad concept of bundling data and methods into a single unit (an object). **Information Hiding** is the specific practice within encapsulation of using the private access modifier to restrict direct access to internal data, exposing only what is necessary through public interfaces.
7. **Inheritance** promotes code reusability by allowing child classes to use existing code from a parent class. It also establishes a clear hierarchical relationship (parent-child) between classes, making the code more organized and reliable.
8. **A Constant Field** is a variable whose value cannot be changed once it has been assigned. In Java, constants are declared using the static and final keywords and are traditionally written in uppercase (e.g., MAX_LIMIT).
9. **Composition** involves a "has-a" relationship where complex behavior is built by assembling different objects, allowing for more flexibility. **Inheritance** involves an "is-a" relationship, creating a strong coupling where a subclass extends the behavior of a superclass.
10. **Parallel Arrays** are two or more arrays where the subscripts relate to each other to store fields of an object. They are considered inferior because they are not easily scalable and lack the integrity and organization provided by a single array of objects.

Part 3: Essay Format Questions

Instructions: These questions require a deeper synthesis of OOP theory and practical application. No answers are provided; use these to practice structuring long-form technical explanations.

1. **Data Integrity and Access Control:** Discuss how the use of private access modifiers combined with accessor and mutator methods ensures the integrity of

an object's state. Compare this approach to a system where all attributes are public.

2. **The Power of Polymorphism:** Explain how polymorphism allows a single array of a superclass type to manage various subclass objects. Provide examples from the traffic enforcement or audition scenarios where this would be advantageous.
3. **Code Maintenance and Reusability:** Compare the advantages and disadvantages of Inheritance versus Composition. Focus on how "coupling" affects a programmer's ability to modify a system without introducing errors in related classes.
4. **Secondary Storage Solutions:** Evaluate the choice between using a delimited text file and a relational database for permanent data storage. Consider factors such as scalability, data integrity (SQL injection), and the amount of code required for processing (sorting/searching).
5. **OOP Design Patterns:** Analyze the role of the toString() method and constructors (default vs. parameterized) in providing a professional interface for class users. Explain why a programmer might choose to overload constructors within a class like Audition or Source.

Part 4: Glossary of Key Terms

Term, Definition

Abstraction, "The process of simplifying complex systems by modeling classes based on essential properties, focusing on what an object does rather than how it does it."

Access Modifier, "Keywords (Public, Private, Protected) that control the visibility and accessibility of classes, fields, and methods."

Application, "A program containing a main method used to instantiate objects and execute logic, as opposed to a class which acts as a blueprint."

Class, "A blueprint or template for creating objects, defining their properties (fields) and behaviors (methods)."

Constructor, "A special method called automatically during object instantiation to initialize the object's state and assign initial values to fields."

Encapsulation, "The methodology of combining data and the methods that operate on that data into a single unit or object."

Extends, "The Java reserved word used in a class header to indicate that a child class is inheriting from a parent class."

Helper Method,A method (also called an auxiliary method) designed to decompose code and perform specific internal tasks for other functions.

Information Hiding,The practice of restricting access to the internal implementation details of an object using private modifiers to prevent misuse.

Inheritance,"A mechanism where a subclass derives properties and behaviors from a superclass, promoting code reuse."

Object,A specific instance of a class that has its own state (attributes) and behavior (methods).

Overloading,Defining multiple methods in the same class with the same name but different parameter lists.

Overriding,Redefining a method in a child class that already exists in the parent class with the identical signature.

Polymorphism,"The ability of a variable or object to take on multiple forms, allowing different object types to be processed through a common interface."

Protected (#),An access modifier that allows a property or method to be accessed only by the class itself or its descendant classes.

Signature,"The combination of a method's name and its parameter list, used by the compiler to distinguish between methods."

Static,A keyword indicating that a field or method belongs to the class itself rather than a specific object instance.

this.,A reserved word used to distinguish between class attributes and parameters when they share the same name.

toString(),A standard typed method used to combine an object's fields into a formatted string for display purposes.