Name: _____

Java Object Oriented Programming (OOP)

What is Object Oriented Programming?

Object Oriented Programming (OOP) is a programming style where you organize code around "objects" that represent real-world things. Instead of writing all your code in one place, you create blueprints (called classes) that describe what objects can do and what information they hold. Each object is an instance of a class.

☐ Simple Analogy: Classes are Like Cookie Cutters

Think of OOP like baking cookies:

- **Cookie Cutter** = Class (the blueprint/template)
  - Defines the shape and design
- **Actual Cookie** = Object (a real instance)
  - Made from the cookie cutter, but is its own individual cookie
- **Cookie Decorations** = Attributes/Properties (data)
  - Each cookie can have different colors, toppings
- ☐☐ **What You Do With Cookies** = Methods (actions)
  - Eat, decorate, package, display

One cookie cutter → Many unique cookies
One class → Many unique objects

Key OOP Concepts

- **Class:** A blueprint/template that defines properties and behaviors
- **Object:** A specific instance created from a class
- **Attributes/Fields:** Variables that store an object's data (what it has)
- **Methods:** Functions that define an object's behavior (what it can do)
- **Constructor:** Special method that creates and initializes objects

Creating a Simple Class

Example 1: Basic Class Structure

```
// Dog.java - The class (blueprint)
public class Dog {
 // Attributes (what a dog has)
 String name;
 int age;
 String breed;

 // Constructor (how to create a dog)
 public Dog(String dogName, int dogAge, String dogBreed) {
  name = dogName;
  age = dogAge;
  breed = dogBreed;
```

```
  }

  // Methods (what a dog can do)
  public void bark() {
    System.out.println(name + " says: Woof! Woof!");
  }

  public void displayInfo() {
    System.out.println("Name: " + name);
    System.out.println("Age: " + age + " years old");
    System.out.println("Breed: " + breed);
  }
}
```

Example 2: Creating and Using Objects

```
// Main.java - Using the Dog class
public class Main {
  public static void main(String[] args) {
    // Create objects (actual dogs) from the Dog class
    Dog dog1 = new Dog("Max", 3, "Golden Retriever");
    Dog dog2 = new Dog("Bella", 5, "Poodle");

    // Use the objects by calling their methods
    dog1.displayInfo();
    dog1.bark();

    System.out.println();

    dog2.displayInfo();
    dog2.bark();
  }
}
```

```
Name: Max
Age: 3 years old
Breed: Golden Retriever
Max says: Woof! Woof!

Name: Bella
Age: 5 years old
Breed: Poodle
Bella says: Woof! Woof!
```

Constructors Explained

What is a Constructor?

A **constructor** is a special method that runs when you create a new object. It has the same name as the class and no return type.

- **Purpose:** Initialize the object's attributes with starting values
- **Name:** Always matches the class name exactly
- **Called:** Automatically when you use the **new** keyword
- **Default:** Java creates a basic constructor if you don't write one

Example 3: Multiple Constructors

```
public class Student {
 String name;
 int grade;
 double gpa;

 // Constructor 1: All parameters
 public Student(String n, int g, double gpaValue) {
  name = n;
  grade = g;
  gpa = gpaValue;
 }

 // Constructor 2: Just name and grade
 public Student(String n, int g) {
  name = n;
  grade = g;
  gpa = 0.0; // Default value
 }

 public void displayInfo() {
  System.out.println("Name: " + name);
  System.out.println("Grade: " + grade);
  System.out.println("GPA: " + gpa);
 }
}

// Using different constructors
Student s1 = new Student("Alice", 11, 3.8);
Student s2 = new Student("Bob", 10);
```

Getters and Setters (Accessors and Mutators)

Example 4: Private Attributes with Getters and Setters

```
public class BankAccount {
 // Private attributes (can't access directly from outside)
 private String accountNumber;
 private double balance;

 public BankAccount(String accNum, double startBalance) {
  accountNumber = accNum;
  balance = startBalance;
 }
```

```
    // Getter - returns the value
    public double getBalance() {
        return balance;
    }

    // Setter - changes the value (with validation)
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited: $" + amount);
        } else {
            System.out.println("Invalid amount!");
        }
    }

    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            System.out.println("Withdrew: $" + amount);
        } else {
            System.out.println("Insufficient funds!");
        }
    }

    public void displayBalance() {
        System.out.println("Account: " + accountNumber);
        System.out.println("Balance: $" + balance);
    }
}
```

Using the BankAccount Class

```
BankAccount myAccount = new BankAccount("12345", 1000.0);

myAccount.displayBalance();
myAccount.deposit(500.0);
myAccount.withdraw(200.0);
myAccount.displayBalance();

// Can't do this! balance is private
// myAccount.balance = 999999; // ERROR!

// Must use getter instead
double currentBalance = myAccount.getBalance();
```

```
Account: 12345
Balance: $1000.0
Deposited: $500.0
Withdrew: $200.0
Account: 12345
Balance: $1300.0
```

The "this" Keyword

Example 5: Using "this" to Avoid Confusion

```
public class Person {
 private String name;
 private int age;

 // "this" refers to the current object's variables
 public Person(String name, int age) {
  this.name = name; // this.name = object's name
  this.age = age; // this.age = object's age
 }

 public void haveBirthday() {
  this.age++; // Increase this object's age
  System.out.println("Happy birthday! Now " + this.age);
 }

 public String getName() {
  return this.name;
 }
}
```

Why use "this"? It makes clear you're referring to the object's variable, not a parameter with the same name.

Complete Real-World Example

Example 6: Book Class with Full OOP Concepts

```
public class Book {
 // Private attributes (encapsulation)
 private String title;
 private String author;
 private int pages;
 private boolean isCheckedOut;

 // Constructor
 public Book(String title, String author, int pages) {
  this.title = title;
  this.author = author;
  this.pages = pages;
  this.isCheckedOut = false;
 }

 // Getters
 public String getTitle() {
  return title;
 }
```

```
 public String getAuthor() {
  return author;
 }

 // Methods (behaviors)
 public void checkOut() {
  if (!isCheckedOut) {
   isCheckedOut = true;
   System.out.println(title + " has been checked out.");
  } else {
   System.out.println(title + " is already checked out.");
  }
 }

 public void returnBook() {
  if (isCheckedOut) {
   isCheckedOut = false;
   System.out.println(title + " has been returned.");
  } else {
   System.out.println(title + " wasn't checked out.");
  }
 }

 public void displayInfo() {
  System.out.println("Title: " + title);
  System.out.println("Author: " + author);
  System.out.println("Pages: " + pages);
  System.out.println("Status: " + (isCheckedOut ? "Checked Out" : "Available"));
 }
}
```

Using the Book Class - Library System

```
public class Library {
 public static void main(String[] args) {
  // Create book objects
  Book book1 = new Book("Harry Potter", "J.K. Rowling", 309);
  Book book2 = new Book("1984", "George Orwell", 328);

  // Display information
  book1.displayInfo();
  System.out.println();

  // Check out a book
  book1.checkOut();
  book1.displayInfo();
  System.out.println();

  // Try to check out again
  book1.checkOut();
  System.out.println();
```

```
    // Return the book
    book1.returnBook();
    book1.displayInfo();
  }
}
```

Key OOP Principles

Principle	What It Means	Example
Encapsulation	Hide internal details, provide controlled access	Private variables with getters/setters
Abstraction	Show only what's necessary, hide complexity	Simple methods hide complicated code
Objects	Create multiple instances from one class	Many Dog objects from one Dog class
Reusability	Write once, use many times	Use the same class in different programs

Benefits of OOP

- **Organization:** Code is structured and easy to understand
- **Reusability:** Classes can be used in multiple programs
- **Maintainability:** Easy to update and fix
- **Real-world modeling:** Represents things naturally
- **Data protection:** Private attributes keep data safe
- **Modularity:** Each class is independent

Key Terms Summary

- **Class:** Blueprint/template
- **Object:** Instance of a class
- **Constructor:** Creates objects
- **Method:** What an object can do
- **Attribute:** What an object has
- **this:** Refers to current object
- **private:** Only accessible inside class
- **public:** Accessible from anywhere

Common OOP Patterns

Pattern 1: Standard Class Template

```
public class ClassName {
  // Private attributes
  private String attribute1;
  private int attribute2;

  // Constructor
  public ClassName(String attr1, int attr2) {
    this.attribute1 = attr1;
    this.attribute2 = attr2;
```

```
    }

    // Getter methods
    public String getAttribute1() {
        return attribute1;
    }

    // Setter methods
    public void setAttribute1(String attr1) {
        this.attribute1 = attr1;
    }

    // Other methods
    public void someMethod() {
        // Method code here
    }
}
```

Quick Practice: Rectangle Class

```
public class Rectangle {
    private double length;
    private double width;

    public Rectangle(double l, double w) {
        length = l;
        width = w;
    }

    public double getArea() {
        return length * width;
    }

    public double getPerimeter() {
        return 2 * (length + width);
    }

    public void displayInfo() {
        System.out.println("Length: " + length);
        System.out.println("Width: " + width);
        System.out.println("Area: " + getArea());
        System.out.println("Perimeter: " + getPerimeter());
    }
}

// Using the Rectangle class
Rectangle rect = new Rectangle(5.0, 3.0);
rect.displayInfo();
```

OOP Best Practices

- **Make attributes private** - Protect your data with encapsulation
- **Use meaningful names** - Class names should be nouns (Dog, Book, Student)
- **One class per file** - Each class should have its own .java file
- **Constructor initializes all attributes** - Set starting values for everything
- **Methods should do one thing** - Keep methods focused and simple
- **Use getters/setters for private attributes** - Control how data is accessed
- **Use "this" when needed** - Makes code clearer

⚠ Common OOP Mistakes

- **Forgetting "new" keyword** - Objects must be created with "new"
- **Public attributes** - Should usually be private for protection
- **No constructor** - Always provide a way to initialize objects
- **Static confusion** - Remember: objects need non-static methods
- **Wrong method calls** - Use objectName.methodName(), not ClassName.methodName()
- **Null pointer errors** - Always create object before using it

Your Turn: Write Your Own Definition

What is Object Oriented Programming? How would you explain it to a friend?

Write your definition in your own words:

Create a simple Car class with attributes and methods:

```
public class Car {
 // Add 3 private attributes (brand, model, year)


 // Add a constructor


 // Add a method to display car information


}
```

Explain the difference between a class and an object:

Why do we make attributes private and use getters/setters?

Give three real-world examples of objects you could model with OOP:

1. _____

2. _____

3. _____