**Lab 5 Tutorial: Advanced Inheritance Concepts**

**Table of Contents**

1. **Static Methods and Variables**
2. **Inheritance with Additional Examples**
3. **Using the Object Class**
4. **The toString() Method**
5. **Comprehensive Program Example**

**1. Static Methods and Variables**

In Java, static variables and methods belong to the class rather than to instances of the class. This means:

* **Static Variables** are shared across all instances of a class.
* **Static Methods** can be called directly on the class without creating an instance.

**Example: Animal Count Tracker**

In this example, we’ll track the number of Animal instances created.

class Animal {

private String name;

private int age;

static int count = 0; // Static variable to keep track of Animal instances

public Animal(String name, int age) {

this.name = name;

this.age = age;

count++; // Increment count for each Animal instance

}

public void eat() {

System.out.println(name + " eats food.");

}

// Static method to get the count of all created Animal objects

public static int getAnimalCount() {

return count;

}

}

**2. Inheritance with Additional Examples**

Inheritance allows classes to reuse code from other classes, forming a hierarchy. Below, we extend Animal to create specific animal types.

**Example: Dog and Cat Classes**

class Dog extends Animal {

public Dog(String name, int age) {

super(name, age);

}

public void bark() {

System.out.println(getName() + " barks.");

}

}

class Cat extends Animal {

public Cat(String name, int age) {

super(name, age);

}

public void meow() {

System.out.println(getName() + " meows.");

}

}

**3. Using the Object Class**

The Object class is the root of the Java class hierarchy. Every class implicitly extends Object. Important methods inherited from Object include:

* **toString()**: Provides a string representation of an object.
* **equals()**: Checks if two objects are equal.

**4. The toString() Method**

By default, toString() in the Object class returns the class name and hashcode. It’s often overridden to provide a meaningful description of the object’s attributes.

**Example: Overriding toString() in Animal Class**

class Animal {

private String name;

private int age;

static int count = 0;

public Animal(String name, int age) {

this.name = name;

this.age = age;

count++;

}

@Override

public String toString() {

return "Animal[name=" + name + ", age=" + age + "]";

}

}

public class Main {

public static void main(String[] args) {

Animal animal1 = new Animal("Lion", 5);

Animal animal2 = new Animal("Tiger", 3);

System.out.println(animal1.toString()); // Displays a customized string for animal1

System.out.println("Total animals: " + Animal.getAnimalCount()); // Using static method

}

}

**5. Comprehensive Program Example**

Below is a program that demonstrates:

* Inheritance with multiple animal classes.
* Static variables and methods to track instances.
* toString() to provide a meaningful object description.

class Animal {

private String name;

private int age;

static int count = 0;

public Animal(String name, int age) {

this.name = name;

this.age = age;

count++;

}

public void eat() {

System.out.println(name + " eats food.");

}

public static int getAnimalCount() {

return count;

}

@Override

public String toString() {

return "Animal[name=" + name + ", age=" + age + "]";

}

}

class Dog extends Animal {

public Dog(String name, int age) {

super(name, age);

}

public void bark() {

System.out.println(getName() + " barks.");

}

@Override

public String toString() {

return super.toString() + " [Dog]";

}

}

class Cat extends Animal {

public Cat(String name, int age) {

super(name, age);

}

public void meow() {

System.out.println(getName() + " meows.");

}

@Override

public String toString() {

return super.toString() + " [Cat]";

}

}

public class Main {

public static void main(String[] args) {

Dog dog1 = new Dog("Buddy", 3);

Cat cat1 = new Cat("Whiskers", 2);

System.out.println(dog1.toString());

System.out.println(cat1.toString());

System.out.println("Total animals created: " + Animal.getAnimalCount());

}

}

**Summary**

* **Static Members**: Useful for class-wide data and functions.
* **Inheritance**: Enables reuse of common properties and methods.
* **Object Class and toString()**: Enhances object descriptions for easier debugging and logging.

**Assignment Description: Simple Inventory Management Application**

**Objective:** Create a simple inventory management application that demonstrates the concepts of inheritance, method overriding, static functions and variables, and user input using the Scanner class. The application will consist of a base class Item and derived classes for specific item types (e.g., Electronics and Groceries).

**Task Breakdown:**

1. **Classes:**
   * **Base Class:**
     + Item
       - Attributes: itemID, itemName, and quantity.
       - Methods:
         * A constructor to initialize the item details.
         * A static variable to keep track of the total number of items in the inventory.
         * A static method to display the total count of items.
         * A method to add stock to the item.
         * A method to display item details.
   * **Derived/Child Classes:**
     + Electronics - Inherits from Item and includes:
       - An additional attribute for warrantyPeriod.
       - An overridden method to add stock, which could log the addition with a message about the warranty period.
     + Groceries - Inherits from Item and includes:
       - An additional attribute for expirationDate.
       - An overridden method to add stock, which checks if the item is expired before allowing stock addition.
2. **User Input:**
   * Use the Scanner class to take input from the user for creating items in the inventory.
   * Prompt the user to enter the type of item (Electronics or Groceries), item ID, item name, and quantity.
   * After item creation, allow users to add stock to an existing item by entering the item ID and the quantity to be added.
3. **Method Overriding:**
   * Ensure that the addStock method is overridden in both derived classes to implement specific behaviors related to each item type.
4. **Static Functions and Variables:**
   * Implement a static function in the Item class to retrieve the total number of items in the inventory.
   * Use a static variable to maintain the count of items added, which should be incremented in the constructor of each item.
5. **Hints:**
   * Start by designing your classes and their relationships (inheritance).
   * Think about how to structure the addStock method for each derived class to include unique functionalities.
   * Consider how user input will flow through the program—after creating an item, what actions can the user take?
   * Ensure to validate user inputs where necessary, such as checking for valid item IDs when adding stock.

**Expected Outcome:** By the end of this assignment, students should have a functional console application that allows users to manage an inventory of different item types while applying the concepts of inheritance, method overriding, and static methods and variables in Java.

**Console Output Example**

Welcome to the Inventory Management System

1. Add Item

2. Add Stock

3. Show Total Items

4. Exit

Please select an option (1-4): 1

Select Item Type:

1. Electronics

2. Groceries

Please enter your choice (1-2): 1

Enter Item ID: E001

Enter Item Name: Laptop

Enter Quantity: 10

Item added successfully! Total items in inventory: 1

------------------------------------------------

1. Add Item

2. Add Stock

3. Show Total Items

4. Exit

Please select an option (1-4): 1

Select Item Type:

1. Electronics

2. Groceries

Please enter your choice (1-2): 2

Enter Item ID: G001

Enter Item Name: Apples

Enter Quantity: 50

Item added successfully! Total items in inventory: 2

------------------------------------------------

1. Add Item

2. Add Stock

3. Show Total Items

4. Exit

Please select an option (1-4): 2

Enter Item ID to add stock: E001

Enter Quantity to add: 5

Stock added successfully! New quantity of Laptop: 15

------------------------------------------------

1. Add Item

2. Add Stock

3. Show Total Items

4. Exit

Please select an option (1-4): 3

Total items in inventory: 2

------------------------------------------------

1. Add Item

2. Add Stock

3. Show Total Items

4. Exit

Please select an option (1-4): 4

Exiting the Inventory Management System. Goodbye!

**Explanation of the Output:**

1. **Welcome Message:** The application starts with a welcome message and a menu of options.
2. **Adding Items:** When the user selects the option to add an item, they choose the type of item, input the item ID, name, and quantity, and receive a confirmation message.
3. **Adding Stock:** The user can then add stock to existing items by entering the item ID and the amount to be added, followed by a success message.
4. **Show Total Items:** The user can view the total number of unique items in the inventory.
5. **Exit Option:** The application ends gracefully with a goodbye message when the user chooses to exit.