**Design Patterns and Principles**

**Exercise 1: Implementing the Singleton Pattern**

Code:

package weekone;

public class singleton {

static class lazysingleton{

private static lazysingleton *instance*;

private lazysingleton() {

System.***out***.println("Lazysingleton created");

}

public static lazysingleton getInstance() {

if(*instance*==null) {

*instance* = new lazysingleton();

}

return *instance*;

}

}

static class threadsafesingleton{

private static threadsafesingleton *instance*;

private threadsafesingleton() {

System.***out***.println("threadsafesingleton created");

}

public static synchronized threadsafesingleton getInstance() {

if(*instance*==null) {

*instance* = new threadsafesingleton();

}

return *instance*;

}

}

static class eagersingleton{

private static final eagersingleton ***instance*** = new eagersingleton();

private eagersingleton() {

System.***out***.println("eagersingleton created");

}

public static eagersingleton getInstance() {

return ***instance***;

}

}

public static void main(String[]args) {

System.***out***.println("Testing Lazy Initialization:");

lazysingleton l1 = lazysingleton.*getInstance*();

lazysingleton l2 = lazysingleton.*getInstance*();

System.***out***.println("Same instance: " + (l1 == l2));

System.***out***.println();

System.***out***.println("Testing Thread-safe Lazy Initialization:");

threadsafesingleton ts1 = threadsafesingleton.*getInstance*();

threadsafesingleton ts2 = threadsafesingleton.*getInstance*();

System.***out***.println("Same instance: " + (ts1 == ts2));

System.***out***.println();

System.***out***.println("Testing Eager Initialization:");

eagersingleton e1 = eagersingleton.*getInstance*();

eagersingleton e2 = eagersingleton.*getInstance*();

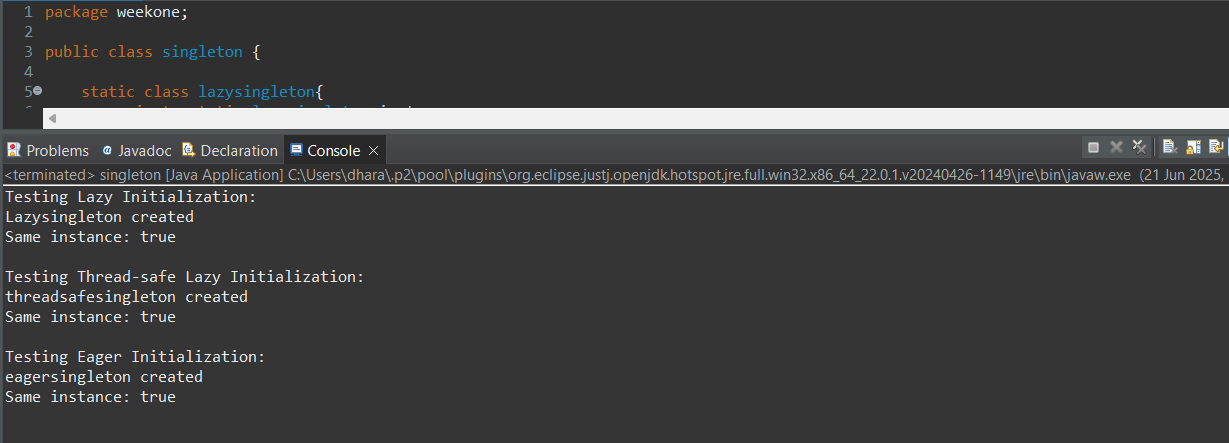
System.***out***.println("Same instance: " + (e1 == e2));

System.***out***.println();

}

}

Output:



**Exercise 2: Implementing the Factory Method Pattern**

Code:

package weekone;

interface animal {

void speak();

}

class dog implements animal {

public void speak() {

System.***out***.println("Dog Barks");

}

}

class cat implements animal{

public void speak() {

System.***out***.println("Cat Meows");

}

}

abstract class animalfactory{

public abstract animal createanimal();

}

class dogfactory extends animalfactory{

public animal createanimal() {

return new dog();

}

}

class catfactory extends animalfactory{

public animal createanimal() {

return new cat();

}

}

public class factorymethodpattern {

public static void main(String[] args) {

animalfactory dogfactory = new dogfactory();

animal dog = dogfactory.createanimal();

dog.speak();

animalfactory catfactory = new catfactory();

animal cat = catfactory.createanimal();

cat.speak();

}

}

Output:

