2/11

**1. Basic Usage of static**

**Question:** Create a class with a static variable and method. Explain how they can be accessed.

**Answer:**

java

public class StaticExample {

static int staticVariable = 10;

static void staticMethod() {

System.out.println("Static method called!");

}

public static void main(String[] args) {

// Accessing static variable

System.out.println("Static Variable: " + StaticExample.staticVariable);

// Accessing static method

StaticExample.staticMethod();

}

}

**Explanation:**

* Static variables and methods belong to the class rather than any instance of the class.
* They can be accessed directly using the class name.

**2. Static Block**

**Question:** Explain the use of a static block with an example.

**Answer:**

java

public class StaticBlockExample {

static int num;

static String str;

// Static block

static {

num = 42;

str = "Hello, Static Block!";

System.out.println("Static block initialized.");

}

public static void main(String[] args) {

System.out.println("Num: " + num);

System.out.println("Str: " + str);

}

}

**Explanation:**

* Static blocks are used to initialize static variables.
* This block gets executed when the class is loaded into memory.

**3. Static Methods and Variables Interaction**

**Question:** How do static methods interact with static variables? Provide an example.

**Answer:**

java

public class InteractionExample {

static int counter = 0;

static void incrementCounter() {

counter++;

}

public static void main(String[] args) {

InteractionExample.incrementCounter();

System.out.println("Counter: " + InteractionExample.counter);

InteractionExample.incrementCounter();

System.out.println("Counter: " + InteractionExample.counter);

}

}

**Explanation:**

* Static methods can access and modify static variables.
* This method can be called without creating an instance of the class.

**4. Singleton Class**

**Question:** Implement a singleton class in Java using the static keyword.

**Answer:**

java

public class Singleton {

private static Singleton instance;

private Singleton() {

// private constructor

}

public static Singleton getInstance() {

if (instance == null) {

instance = new Singleton();

}

return instance;

}

public void displayMessage() {

System.out.println("Singleton instance method called!");

}

public static void main(String[] args) {

Singleton singleton = Singleton.getInstance();

singleton.displayMessage();

}

}

**Explanation:**

* The singleton pattern ensures only one instance of the class is created.
* The static getInstance method provides a global access point to this instance.

**5. Static with Inner Class**

**Question:** Explain static inner classes with an example.

**Answer:**

java

public class OuterClass {

static int outerStaticVar = 5;

static class InnerClass {

void display() {

System.out.println("Outer static variable: " + outerStaticVar);

}

}

public static void main(String[] args) {

OuterClass.InnerClass inner = new OuterClass.InnerClass();

inner.display();

}

}

**Explanation:**

* Static inner classes can access static members of the outer class.
* They do not need an instance of the outer class to be instantiated.

**6. Static Methods and Non-static Variables**

**Question:** Can static methods access non-static variables? Justify with an example.

**Answer:**

java

public class StaticAccessExample {

int nonStaticVar = 10;

static int staticVar = 20;

static void staticMethod() {

// nonStaticVar cannot be accessed here

System.out.println("Static variable: " + staticVar);

}

public void nonStaticMethod() {

System.out.println("Non-static variable: " + nonStaticVar);

System.out.println("Static variable: " + staticVar);

}

public static void main(String[] args) {

StaticAccessExample example = new StaticAccessExample();

example.nonStaticMethod();

staticMethod();

}

}

**Explanation:**

* Static methods cannot directly access non-static variables.
* Non-static methods can access both static and non-static variables.

**7. Static Variables in Different Instances**

**Question:** Demonstrate the behavior of static variables across different instances.

**Answer:**

java

public class InstanceExample {

static int staticVar = 0;

int nonStaticVar = 0;

public void incrementVars() {

staticVar++;

nonStaticVar++;

}

public static void main(String[] args) {

InstanceExample obj1 = new InstanceExample();

InstanceExample obj2 = new InstanceExample();

obj1.incrementVars();

obj2.incrementVars();

System.out.println("obj1 - Static Var: " + obj1.staticVar + ", Non-Static Var: " + obj1.nonStaticVar);

System.out.println("obj2 - Static Var: " + obj2.staticVar + ", Non-Static Var: " + obj2.nonStaticVar);

}

}

**Explanation:**

* Static variables are shared among all instances of a class.
* Non-static variables are instance-specific.

**8. Limitations of Static Methods**

**Question:** List and demonstrate limitations of static methods.

**Answer:**

java

public class LimitationsExample {

int instanceVar = 10;

static void staticMethod() {

// instanceVar cannot be accessed here

System.out.println("Cannot access instance variables directly.");

}

public void instanceMethod() {

System.out.println("Instance Variable: " + instanceVar);

}

public static void main(String[] args) {

LimitationsExample example = new LimitationsExample();

example.instanceMethod();

staticMethod();

}

}

**Explanation:**

* Static methods cannot access instance variables or instance methods.
* They can only interact with static variables and methods.

**9. Static Keyword with Arrays**

**Question:** How can the static keyword be used with arrays? Provide an example.

**Answer:**

java

public class StaticArrayExample {

static int[] numbers = {1, 2, 3, 4, 5};

static void displayNumbers() {

for (int num : numbers) {

System.out.print(num + " ");

}

}

public static void main(String[] args) {

StaticArrayExample.displayNumbers();

}

}

**Explanation:**

* Static arrays can be shared across all instances of the class.
* Static methods can manipulate these arrays.

**10. Static Keyword in Abstract Classes**

**Question:** Can the static keyword be used in abstract classes? Illustrate with an example.

**Answer:**

java

abstract class AbstractClass {

static int staticVar = 5;

static void staticMethod() {

System.out.println("Static method in abstract class");

}

abstract void instanceMethod();

}

public class SubClass extends AbstractClass {

void instanceMethod() {

System.out.println("Implementation of abstract method");

}

public static void main(String[] args) {

AbstractClass.staticMethod();

SubClass sub = new SubClass();

sub.instanceMethod();

}

}

**Explanation:**

* Static methods and variables can be defined in abstract classes.
* They can be accessed without creating an instance of the abstract class.

**11. Static Methods Calling Non-static Methods**

**Question:** How can a static method call a non-static method? Provide an example.

**Answer:**

java

public class StaticNonStaticExample {

void nonStaticMethod() {

System.out.println("Non-static method called!");

}

static void staticMethod() {

StaticNonStaticExample example = new StaticNonStaticExample();

example.nonStaticMethod();

}

public static void main(String[] args) {

staticMethod();

}

}

**Explanation:**

* Static methods can call non-static methods by creating an instance of the class.
* This allows access to instance methods and variables.

**12. Static Methods in Interfaces**

**Question:** Can static methods be declared in interfaces? Provide an example.

**Answer:**

java

interface StaticInterface {

static void staticMethod() {

System.out.println("Static method in interface");

}

}

public class InterfaceExample implements StaticInterface {

public static void main(String[] args) {

StaticInterface.staticMethod();

}

}

**Explanation:**

* Static methods can be declared in interfaces.
* They can be called using the interface name.

**13. Static Import**

**Question:** Explain the use of static import with an example.

**Answer:**

java

import static java.lang.Math.\*;

public class StaticImportExample {

public static void main(String[] args) {

System.out.println("Square root of 16: " + sqrt(16));

System.out.println("Value of PI: " + PI);

}

}

**Explanation:**

* Static import allows the static members of a class to be used without class qualification.
* It makes the code more readable and concise.

**14. Static Final Variables**

**Question:** What are static final variables? Provide an example.

**Answer:**

java

public class StaticFinalExample {

continue generatinh

Sure! Let's continue with the remaining Java coding questions:

**14. Static Final Variables**

**Question:** What are static final variables? Provide an example.

**Answer:**

java

public class StaticFinalExample {

static final int CONSTANT = 100;

public static void main(String[] args) {

// CONSTANT cannot be changed, uncommenting below line will cause an error

// CONSTANT = 200;

System.out.println("Static final variable: " + CONSTANT);

}

}

**Explanation:**

* Static final variables are constants that cannot be changed once assigned.
* They are shared among all instances and remain constant.

**15. Static Methods in Inheritance**

**Question:** Can static methods be overridden in inheritance? Provide an example.

**Answer:**

java

class ParentClass {

static void staticMethod() {

System.out.println("Static method in ParentClass");

}

}

class ChildClass extends ParentClass {

static void staticMethod() {

System.out.println("Static method in ChildClass");

}

}

public class InheritanceExample {

public static void main(String[] args) {

ParentClass.staticMethod();

ChildClass.staticMethod();

}

}

**Explanation:**

* Static methods cannot be overridden; they are hidden.
* The static method in the subclass hides the static method in the superclass.

**16. Static Import with Custom Classes**

**Question:** Demonstrate the use of static import with custom classes.

**Answer:**

java

package mypackage;

public class Utility {

public static void printMessage(String message) {

System.out.println(message);

}

}

// Main class

import static mypackage.Utility.\*;

public class StaticImportCustomExample {

public static void main(String[] args) {

printMessage("Hello from static import!");

}

}

**Explanation:**

* Static import can be used with custom classes to import static methods.
* It simplifies code usage.

**17. Static Nested Interface**

**Question:** Explain static nested interfaces with an example.

**Answer:**

java

public class OuterClass {

interface StaticNestedInterface {

void display();

}

static class InnerClass implements StaticNestedInterface {

public void display() {

System.out.println("Static nested interface method implemented in InnerClass");

}

}

public static void main(String[] args) {

StaticNestedInterface obj = new InnerClass();

obj.display();

}

}

**Explanation:**

* Interfaces can be nested within a class as static.
* They can be implemented by static inner classes.

**18. Static Members in Enumeration**

**Question:** How can static members be used in enumeration? Provide an example.

**Answer:**

java

public enum Day {

MONDAY, TUESDAY, WEDNESDAY;

static String description = "Days of the week";

public static void printDescription() {

System.out.println(description);

}

public static void main(String[] args) {

Day.printDescription();

}

}

**Explanation:**

* Static members can be declared in an enumeration.
* They can be accessed using the enumeration name.

**19. Static Utility Class**

**Question:** Create a static utility class and demonstrate its usage.

**Answer:**

java

public class MathUtils {

private MathUtils() {

// private constructor to prevent instantiation

}

public static int add(int a, int b) {

return a + b;

}

public static int subtract(int a, int b) {

return a - b;

}

}

public class UtilityClassExample {

public static void main(String[] args) {

System.out.println("Addition: " + MathUtils.add(5, 3));

System.out.println("Subtraction: " + MathUtils.subtract(5, 3));

}

}

**Explanation:**

* Static utility classes have private constructors to prevent instantiation.
* They provide static methods for common operations.

**20. Static Members in Annotations**

**Question:** Can static members be used in annotations? Provide an example.

**Answer:**

java

import java.lang.annotation.\*;

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.METHOD)

@interface Test {

String description() default "No description";

static String author = "Author Name";

}

public class AnnotationExample {

@Test(description = "Example method")

public void exampleMethod() {

System.out.println("Example method executed");

}

public static void main(String[] args) {

Test annotation = AnnotationExample.class.getMethods()[0].getAnnotation(Test.class);

System.out.println("Description: " + annotation.description());

System.out.println("Author: " + Test.author);

}

}

**Explanation:**

* Static members can be used in annotations.
* They provide additional information for the annotation.

**Singleton Class Questions**

**1. Basic Singleton Class**

**Question:** Implement a basic singleton class in Java.

**Answer:**

java

public class BasicSingleton {

private static BasicSingleton instance;

private BasicSingleton() {

// private constructor

}

public static BasicSingleton getInstance() {

if (instance == null) {

instance = new BasicSingleton();

}

return instance;

}

}

**Explanation:**

* The basic singleton class ensures only one instance is created.
* The getInstance method provides access to the instance.

**2. Thread-safe Singleton Class**

**Question:** Implement a thread-safe singleton class in Java.

**Answer:**

java

public class ThreadSafeSingleton {

private static ThreadSafeSingleton instance;

private ThreadSafeSingleton() {

// private constructor

}

public static synchronized ThreadSafeSingleton getInstance() {

if (instance == null) {

instance = new ThreadSafeSingleton();

}

return instance;

}

}

**Explanation:**

* The synchronized method ensures thread safety.
* Only one instance is created, even in a multi-threaded environment.

**3. Eager Initialization Singleton Class**

**Question:** Implement an eager initialization singleton class in Java.

**Answer:**

java

public class EagerSingleton {

private static final EagerSingleton instance = new EagerSingleton();

private EagerSingleton() {

// private constructor

}

public static EagerSingleton getInstance() {

return instance;

}

}

**Explanation:**

* The instance is created at class loading time.
* This approach ensures thread safety.

**4. Static Block Initialization Singleton Class**

**Question:** Implement a singleton class using static block initialization in Java.

**Answer:**

java

public class StaticBlockSingleton {

private static final StaticBlockSingleton instance;

static {

try {

instance = new StaticBlockSingleton();

} catch (Exception e) {

throw new RuntimeException("Exception occurred in creating singleton instance");

}

}

private StaticBlockSingleton() {

// private constructor

}

public static StaticBlockSingleton getInstance() {

return instance;

}

}

**Explanation:**

* Static block initialization allows for exception handling.
* The instance is created at class loading time.

**5. Bill Pugh Singleton Class**

**Question:** Implement the Bill Pugh singleton design in Java.

**Answer:**

java

public class BillPughSingleton {

private BillPughSingleton() {

// private constructor

}

private static class SingletonHelper {

private static final BillPughSingleton INSTANCE = new BillPughSingleton();

}

public static BillPughSingleton getInstance() {

return SingletonHelper.INSTANCE;

}

}

**Explanation:**

* This approach uses a static inner helper class.
* The instance is created when the inner class is loaded.

**6. Enum Singleton Class**

**Question:** Implement a singleton class using an enum in Java.

**Answer:**

java

public enum EnumSingleton {

INSTANCE;

public void displayMessage() {

System.out.println("Enum Singleton instance called!");

}

}

public class EnumSingletonExample {

public static void main(String[] args) {

EnumSingleton singleton = EnumSingleton.INSTANCE;

singleton.displayMessage();

}

}

**Explanation:**

* Enums provide a simple and thread-safe way to implement singletons.
* They also prevent issues with reflection and serialization.

**7. Reflection Safe Singleton Class**

**Question:** Implement a reflection-safe singleton class in Java.

**Answer:**

java

public class ReflectionSafeSingleton {

private static ReflectionSafeSingleton instance;

private ReflectionSafeSingleton() {

if (instance != null) {

throw new IllegalStateException("Instance already created");

}

}

public static ReflectionSafeSingleton getInstance() {

if (instance == null) {

instance = new ReflectionSafeSingleton();

}

return instance;

}

}

**Explanation:**

* The constructor throws an exception if an instance already exists.
* This prevents creating a new instance through reflection.

**8. Serialization Safe Singleton Class**

**Question:** Implement a serialization-safe singleton class in Java.

**Answer:**

java

import java.io.Serializable;

public class SerializationSafeSingleton implements Serializable {

private static final long serialVersionUID = 1L;

private static SerializationSafeSingleton instance;

private SerializationSafeSingleton() {

// private constructor

}

public static SerializationSafeSingleton getInstance() {

if (instance == null) {

instance = new SerializationSafeSingleton();

}

return instance;

}

// Ensure that during serialization we only return one instance

protected Object readResolve() {

return getInstance();

}

}

**Explanation:**

* The readResolve method ensures that the same instance is returned during