

Java Constructor I

- ❶ **Purpose:** A constructor is a special block of code used to **initialize a newly created object**. Its primary job is to put the object into a valid initial state.
- ❷ **Not a Method:** Although it looks like a method, a constructor is **not** a method. It doesn't have a return type, not even `void`.
- ❸ **Name:** The constructor **must have the exact same name** as the class it is declared in.
- ❹ **Automatic Call:** A constructor is automatically invoked at the time of object creation using the `new` keyword (e.g., `MyClass obj = new MyClass();`).
- ❺ **No Return Type:** You cannot specify any return type (like `int`, `String`, or `void`) for a constructor. If you do, it becomes a regular method.

Java Constructor II

- ⑥ **Default Constructor:** If a class does not explicitly define *any* constructor, the Java compiler automatically provides a no-argument constructor called the **default constructor**. It initializes member variables to their default values (e.g., `0`, `null`, `false`).
- ⑦ **No-Argument Constructor:** A constructor that takes no parameters. It can be provided by the programmer or by the compiler (as the default constructor).
- ⑧ **Parameterized Constructor:** A constructor that accepts parameters. It is used to provide different initial values to different objects when they are created.
- ⑨ **Constructor Overloading:** A class can have multiple constructors with different parameter lists. This is known as constructor overloading. The correct constructor is chosen based on the arguments provided during object creation.

Java Constructor III

- ⑩ **Inheritance and super():** The first statement inside any constructor is an implicit call to the constructor of its immediate parent class using `super()`. This ensures the parent part of the object is built first.
- ⑪ **Explicit super() call:** You can explicitly write `super(...)` to call a specific parent constructor, but it **must be the first statement** in the child constructor.
- ⑫ **Chaining with this():** A constructor can call another constructor within the same class using `this(...)`. This is used to avoid code duplication and **must also be the first statement**.
- ⑬ **No simultaneous this() and super():** Since both `this()` and `super()` must be the first statement, a constructor cannot have both.

Java Constructor IV

- ⑭ **No Inheritance:** Constructors are **not inherited** by subclasses. A subclass must define its own constructors, which will chain to a parent constructor.
- ⑮ **Access Modifiers:** Constructors can have access modifiers (**public**, **protected**, **private**, or package-private). A **private** constructor is used to prevent instantiation from outside the class (e.g., in utility classes or Singleton design pattern).
- ⑯ **Not Static:** Constructors are inherently associated with object creation and are **not static**. They can only be called on an instance (via **new**).
- ⑰ **No Abstract or Final:** Constructors cannot be declared as **abstract**, **static**, or **final**.
- ⑱ **Can Throw Exceptions:** Constructors can throw exceptions, which means object creation might fail if the constructor throws an exception.

Java Constructor V

- 19 **Copy Constructor:** While not built-in, you can create a constructor that takes an object of the same class as a parameter and copies its state. This is used to create a copy of an object.
- 20 **Private Constructor for Singleton:** A class with a `private` constructor and a `static` method that returns the same instance is the standard way to implement the **Singleton pattern**.
- 21 **Constructor in Abstract Class:** **Abstract classes can have constructors.** They are called when a concrete subclass is instantiated, helping to initialize the state of the abstract part of the object.

Java Constructor VI

- 22 **Default Constructor Disappears:** If you define *any* constructor (e.g., a parameterized one) in a class, the Java compiler **does not provide the default no-arg constructor** anymore. You must define it explicitly if needed.
- 23 **Implicit Super Call Fails:** If a parent class does not have a no-argument constructor, the child class constructor **must explicitly** call a parameterized parent constructor using `super(...)`.
- 24 **Initialization Blocks:** Instance initializer blocks run before the constructor and are copied into every constructor by the compiler. They are useful for code shared by all constructors.
- 25 **Order of Execution:** For a class, the order is:
 - Static variables and static blocks (once when class is loaded).

Java Constructor VII

- Instance variables and instance initialization blocks (every object creation).
 - The body of the constructor.
- 26 **Implicit Return:** Even though you can't specify a return type, a constructor **implicitly returns the instance** of the class (the newly created object).
 - 27 **Anonymous Classes:** Anonymous classes cannot have explicit constructors. Their initialization is done via instance initializer blocks.
 - 28 **Reflection:** Constructors can be accessed and invoked at runtime using Java's **Reflection API** ([java.lang.reflect.Constructor](#) class).
 - 29 **Generic Classes:** Constructors for generic classes do not require the type parameter. You write `public MyClass()` not `public MyClass<T>()`.

Java Constructor VIII

- ❿ **Performance:** Constructor calls are generally fast, but heavy initialization logic inside them can impact performance during object creation.

Java Default Constructor:

```
1 <class_name>()  
2 {  
3  
4 }
```

What is the purpose of a default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Java Constructor IX

```
1 //Let us see another example of default constructor which displays the default values
2 class Student3
3 {
4     int id;
5     String name;
6     //method to display the value of id and name
7     void display()
8     {
9         System.out.println(id+" "+name);
10    }
11    public static void main(String args[])
12    {
13        //creating objects
14        Student3 s1=new Student3();
15        Student3 s2=new Student3();
16        s1.display();
17        s2.display();
18    }
19 }
20 Output:
21 0 null
22 0 null
23 In the above class,you are not creating any constructor so compiler provides you a default constructor.
```



Java Constructor X

24 Here 0 and null values are provided by default constructor.

Rule: If there is no constructor in a class, compiler automatically creates a default constructor.

Java No-Arg Constructor:

```
1 //Java Program to create and call a default constructor
2 class Bike1
3 {
4     //creating a default constructor
5     Bike1()
6     {
7         System.out.println("Bike is created");
8     }
9     //main method
10    public static void main(String args[])
11    {
12        //calling a default constructor
13        Bike1 b=new Bike1();
14    }
```

Java Constructor XI

```
15 }
1 public class Main
2 {
3     private String name;
4
5     // constructor
6     Main()
7     {
8         System.out.println("Constructor Called:");
9         name = "Programiz";
10    }
11
12    public static void main(String[] args)
13    {
14
15        // constructor is invoked while
16        // creating an object of the Main class
17        Main obj = new Main();
18        System.out.println("The name is " + obj.name);
19    }
20 }
```



Java Constructor XII

```
1 public class Main
2 {
3
4     int i;
5
6     // constructor with no parameter
7     private Main()
8     {
9         i = 5;
10        System.out.println("Constructor is called");
11    }
12
13    public static void main(String[] args)
14    {
15
16        // calling the constructor without any parameter
17        Main obj = new Main();
18        System.out.println("Value of i: " + obj.i);
19    }
20 }
```

Java Constructor XIII

Java Parameterized Constructor:

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

```
1 //Java Program to demonstrate the use of the parameterized constructor.
2 class Student4
3 {
4     int id;
5     String name;
6     //creating a parameterized constructor
7     Student4(int i,String n)
8     {
9         id = i;
10        name = n;
11    }
12    void display()
```



Java Constructor XIV

```
13     {  
14         System.out.println(id+ " "+name);  
15     }  
16     public static void main(String args[])  
17     {  
18         //creating objects and passing values  
19         Student4 s1 = new Student4(111,"Karan");  
20         Student4 s2 = new Student4(222,"Aryan");  
21         s1.display();  
22         s2.display();  
23     }  
24 }
```

Java Constructor XV

Constructor Overloading in Java

- ✓ In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.
- ✓ Constructor overloading in Java is a technique of having more than one constructor with different parameter lists.
- ✓ They are arranged in a way that each constructor performs a different task.
- ✓ They are differentiated by the compiler by the number of parameters in the list and their types.

Example of Constructor Overloading

Java Constructor XVI

```
1 //Java program to overload constructors
2 public class Student5
3 {
4     int id;
5     String name;
6     int age;
7     //creating two arg constructor
8     Student5(int i,String n)
9     {
10         id = i;
11         name = n;
12     }
13     //creating three arg constructor
14     Student5(int i,String n,int a)
15     {
16         id = i;
17         name = n;
18         age=a;
19     }
20     void display()
21     {
22         System.out.println(id+ " "+name+" "+age);
23     }
```

Java Constructor XVII

```
24     public static void main(String args[])
25     {
26         Student5 s1 = new Student5(111, "Karan");
27         Student5 s2 = new Student5(222, "Aryan", 25);
28         s1.display();
29         s2.display();
30     }
31 }
32 Output:
33
34 111 Karan 0
35 222 Aryan 25
```

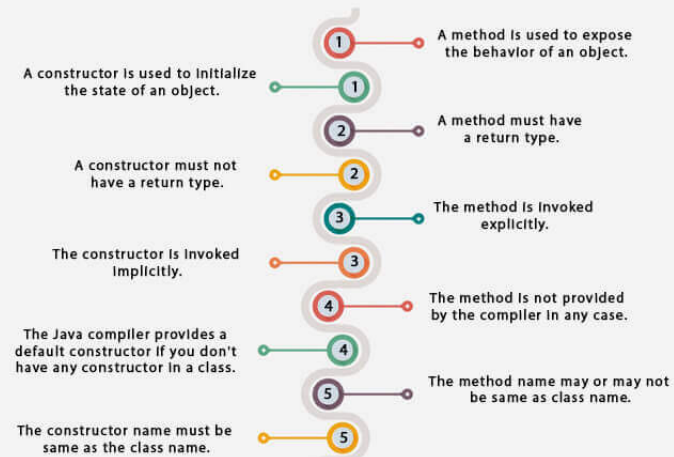
Java Constructor XVIII

There are many differences between constructors and methods.

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

Java Constructor XIX

Difference between constructor and method in Java



Java Constructor Examples I

Question 1: Constructor Chaining

```
1 class Test
2 {
3     Test()
4     {
5         System.out.print("A ");
6     }
7     Test(int x)
8     {
9         this();
10        System.out.print("B ");
11    }
12    public static void main(String[] args)
13    {
14        new Test(10);
15    }
16 }
```

What is the output?

① A B

Java Constructor Examples II

- ② B A
- ③ A
- ④ Compilation error

Correct Answer: A) A B

Explanation: `this()` calls the no-argument constructor first, printing "A", then the current constructor continues executing, printing "B".

Java Constructor Examples III

Question 2: Inheritance Issue

```
1 class Base
2 {
3     Base(String s)
4     {
5         System.out.print("Base: " + s);
6     }
7 }
8 class Derived extends Base
9 {
10     Derived()
11     {
12         System.out.print("Derived");
13     }
14 }
```

What happens when this code compiles?

- ❶ Compiles successfully
- ❷ Runtime error

Java Constructor Examples IV

- ③ Compilation error - implicit super() not available
- ④ Prints "Base: null Derived"

Correct Answer: C) Compilation error - implicit super() not available

Explanation: The Base class has a parameterized constructor but no default constructor. The Derived class constructor implicitly calls `super()`, which doesn't exist.

Java Constructor Examples V

Question 3: Private Constructor

```
1 class Mystery
2 {
3     private Mystery()
4     {
5         System.out.print("Private ");
6     }
7     public static Mystery create()
8     {
9         return new Mystery();
10    }
11    public static void main(String[] args)
12    {
13        Mystery m = Mystery.create();
14    }
15 }
```

What is the output?

- ① Private
- ② Nothing

Java Constructor Examples VI

- ③ Compilation error
- ④ Runtime error

Correct Answer: A) Private

Explanation: Private constructors can be called from within the same class. The static factory method `create()` can instantiate the class.

Java Constructor Examples VII

Question 4: Constructor Call Count

```
1 class A
2 {
3     A()
4     {
5         System.out.print("A ");
6     }
7 }
8 class B extends A
9 {
10     B()
11     {
12         System.out.print("B ");
13     }
14     B(int x)
15     {
16         this(); System.out.print("B" + x);
17     }
18 }
19 // Code: new B(5);
```

How many constructor calls occur?

Java Constructor Examples VIII

- ❶ 1
- ❷ 2
- ❸ 3
- ❹ 4

Correct Answer: B) 2

Explanation: `B(int) → B() → A()` (implicit `super()`). Total 2 explicit constructor calls, but 3 constructors execute.

Java Constructor Examples IX

Question 5: Initialization Order

```
1 class Tricky
2 {
3     static
4     {
5         System.out.print("Static ");
6     }
7     {
8         System.out.print("Init ");
9     }
10    Tricky()
11    {
12        System.out.print("Constructor ");
13    }
14    public static void main(String[] args)
15    {
16        new Tricky();
17    }
18 }
```

What is the output?

Java Constructor Examples X

- ❶ Static Init Constructor
- ❷ Init Static Constructor
- ❸ Static Constructor Init
- ❹ Constructor Init Static

Correct Answer: A) Static Init Constructor

Explanation: Execution order: Static block → Instance initializer → Constructor.

Java Constructor Examples XI

Question 6: Valid Constructor Syntax **Which constructor declaration is valid?**

- ❶ `void Constructor() {}`
- ❷ `static Constructor() {}`
- ❸ `final Constructor() {}`
- ❹ `private Constructor() {}`

Correct Answer: D) `private Constructor() {}`

Explanation: Constructors cannot have return types, cannot be static or final. Private constructors are valid and used for Singleton pattern.

Java Constructor Examples XII

Question 7: Explicit super()

```
1 class Parent
2 {
3     Parent()
4     {
5         System.out.print("P ");
6     }
7 }
8 class Child extends Parent
9 {
10     Child()
11     {
12         super();
13         System.out.print("C ");
14     }
15     public static void main(String[] args)
16     {
17         new Child();
18     }
19 }
```

What is the output?

Java Constructor Examples XIII

- ❶ P C
- ❷ C P
- ❸ P
- ❹ Compilation error

Correct Answer: A) P C

Explanation: Explicit `super()` call works the same as implicit one - parent constructor executes first.

Java Constructor Examples XIV

Question 8: Recursive Constructor

```
1 class Test
2 {
3     Test()
4     {
5         this(10);
6         System.out.print("A ");
7     }
8     Test(int x)
9     {
10        this();
11        System.out.print("B ");
12    }
13 }
```

What happens?

- ❶ Compiles successfully
- ❷ Runtime error
- ❸ Compilation error - recursive constructor invocation

Java Constructor Examples XV

④ Prints "A B"

Correct Answer: C) Compilation error - recursive constructor invocation

Explanation: `this(10)` and `this()` create mutual recursion that's detected at compile time.

Java Constructor Examples XVI

Question 9: Object Counting

```
1 class Counter
2 {
3     static int count = 0;
4     {
5         count++;
6     }
7     Counter()
8     {
9
10    }
11    Counter(int x)
12    {
13        this();
14    }
15    public static void main(String[] args)
16    {
17        new Counter();
18        new Counter(10);
19        System.out.print(count);
20    }
21 }
```



Java Constructor Examples XVII

How many objects are created?

- ❶ 1
- ❷ 2
- ❸ 3
- ❹ 0

Correct Answer: B) 2

Explanation: Instance initializer runs for each object creation, incrementing count for both objects.

Java Constructor Examples XVIII

Question 10: Variable Shadowing

```
1  class X
2  {
3      String s = "Instance";
4      X()
5      {
6          System.out.print(s);
7          String s = "Local";
8          System.out.print(s);
9      }
10     public static void main(String[] args)
11     {
12         new X();
13     }
14 }
```

What is the output?

- ① InstanceLocal
- ② LocalInstance

Java Constructor Examples XIX

- ③ Instance
- ④ Local

Correct Answer: A) InstanceLocal

Explanation: Instance variable accessed first, then local variable shadows it in the same scope.

Java Constructor Examples XX

Question 11: Constructor Characteristics **Which statement is true about constructors?**

Constructors can be inherited

Constructors can be overridden

Constructors can be overloaded

Constructors can be static

Correct Answer: C) Constructors can be overloaded

Explanation: Overloading is having multiple constructors with different parameters. Constructors cannot be inherited, overridden, or static.

Java Constructor Examples XXI

Question 12: Static Initialization Order

```
1 class Loop
2 {
3     static Loop obj = new Loop();
4     static int x = 5;
5     Loop()
6     {
7         System.out.print(x + " ");
8     }
9     public static void main(String[] args)
10    {
11        new Loop();
12    }
13 }
```

What is the output?

5 5

0 5

Java Constructor Examples XXII

0 0

5 0

Correct Answer: B) 0 5

Explanation: Static variable initialization happens after static object creation, so first constructor sees default value 0, second sees initialized value 5.

Java Constructor Examples XXIII

Question 13: Abstract Class Constructor

```
1 abstract class Abstract
2 {
3     Abstract()
4     {
5         System.out.print("Abstract ");
6     }
7 }
8 class Concrete extends Abstract
9 {
10     Concrete()
11     {
12         System.out.print("Concrete ");
13     }
14     public static void main(String[] args)
15     {
16         new Concrete();
17     }
18 }
```

What happens?

Java Constructor Examples XXIV

Abstract Concrete

Concrete Abstract

Compilation error

Runtime error

Correct Answer: A) Abstract Concrete

Explanation: Abstract classes can have constructors that are called when concrete sub-classes are instantiated.

Java Constructor Examples XXV

Question 14: Singleton Pattern Access

```
1 class Secret
2 {
3     private Secret()
4     {
5
6     }
7     public static Secret getInstance()
8     {
9         return new Secret();
10    }
11    void show()
12    {
13        System.out.print("Secret");
14    }
15 }
16 // In another class:
17 Secret s = Secret.getInstance();
18 s.show();
```

What is the output?

Java Constructor Examples XXVI

Secret

Compilation error

Runtime error

Nothing

Correct Answer: A) Secret

Explanation: Private constructor is accessible within the same class, and public static method provides controlled access (Singleton pattern).

Java Constructor Examples XXVII

Question 15: Multi-level Inheritance

```
1 class One
2 {
3     One()
4     {
5         System.out.print("1 ");
6     }
7 }
8 class Two extends One
9 {
10     Two()
11     {
12         System.out.print("2 ");
13     }
14 }
15 class Three extends Two
16 {
17     Three()
18     {
19         System.out.print("3 ");
20     }
21     Three(int x)
22     {
```

Java Constructor Examples XXVIII

```
23         this(); System.out.print("3" + x);
24     }
25 }
26 // Code: new Three(5);
```

How many constructor calls occur?

- 1
- 2
- 3
- 4

Correct Answer: C) 3

Explanation: Three(int) → Three() → Two() → One(). Total 3 constructor executions in the chain.

Java Constructor Examples XXIX

Question 16: Instance Initializer Order

```
1 class InitOrder
2 {
3     {
4         System.out.print("1 ");
5     }
6     InitOrder()
7     {
8         System.out.print("2 ");
9     }
10    {
11        System.out.print("3 ");
12    }
13    public static void main(String[] args)
14    {
15        new InitOrder();
16    }
17 }
```

What is the output?

1 2 3

Java Constructor Examples XXX

1 3 2

2 1 3

3 1 2

Correct Answer: B) 1 3 2

Explanation: All instance initializers run in order of appearance before the constructor body executes.

Java Constructor Examples XXXI

Question 17: Return Statement in Constructor **Which constructor will compile?**

```
class A  A()  return this;
```

```
class B  B()  return null;
```

```
class C  C()  super(); return;
```

```
class D  D()  return new D();
```

Explanation: Constructors can have empty return statements but cannot return values or objects.

Java Constructor Examples XXXII

Question 18: Default Values

```
1 class DefaultTest
2 {
3     int x;
4     DefaultTest()
5     {
6         System.out.print(x);
7     }
8     public static void main(String[] args)
9     {
10         new DefaultTest();
11     }
12 }
```

What is the output?

0

null

Compilation error

Java Constructor Examples XXXIII

Garbage value

Correct Answer: A) 0

Explanation: Instance variables are automatically initialized to default values before constructor execution.

Java Constructor Examples XXXIV

Question 19: Singleton Instantiation

```
1 class Singleton
2 {
3     private static Singleton instance;
4     private Singleton()
5     {
6
7     }
8     public static Singleton getInstance()
9     {
10         if(instance == null)
11         {
12             instance = new Singleton();
13         }
14         return instance;
15     }
16 }
17 // Code: Singleton s1 = new Singleton();
```

What happens?

Creates singleton instance

Java Constructor Examples XXXV

Compilation error

Runtime error

Creates multiple instances

Correct Answer: B) Compilation error

Explanation: Private constructor cannot be accessed from outside the class, preventing direct instantiation.

Java Constructor Examples XXXVI

Question 20: Constructor Exception Handling

```
1 class ExceptionTest
2 {
3     ExceptionTest() throws Exception
4     {
5         throw new Exception();
6     }
7     public static void main(String[] args)
8     {
9         try
10        {
11            new ExceptionTest();
12        }
13        catch(Exception e)
14        {
15            System.out.print("Caught ");
16        }
17    }
18 }
```

What is the output?

Java Constructor Examples XXXVII

Caught

Runtime error

Compilation error

No output

Correct Answer: A) Caught

Explanation: Constructors can throw exceptions, which must be handled by the caller using try-catch.

Java Constructor Examples XXXVIII

Question 21: Object Creation Methods

```
1 class Create
2 {
3     Create()
4     {
5
6     }
7     public static Create getCreate()
8     {
9         return new Create();
10    }
11 }
```

How many ways to create object?

- 1
- 2
- 3

Java Constructor Examples XXXIX

4

Correct Answer: B) 2

Explanation: Direct instantiation: `new Create()` and factory method: `Create.getCreate()`

Java Constructor Examples XL

Question 22: Constructor Chaining Order

```
1 class Chaining
2 {
3     Chaining()
4     {
5         this(10); System.out.print("1 ");
6     }
7     Chaining(int x)
8     {
9         this(10, 20); System.out.print("2 ");
10    }
11    Chaining(int x, int y)
12    {
13        System.out.print("3 ");
14    }
15    public static void main(String[] args)
16    {
17        new Chaining();
18    }
19 }
```

What is the output?

Java Constructor Examples XLI

1 2 3

3 2 1

3

1

Correct Answer: B) 3 2 1

Explanation: Constructor chaining executes deepest constructor first: `Chaining(int,int)`
→ `Chaining(int)` → `Chaining()`.

Java Constructor Examples XLII

Question 23: Default Constructor Facts **Which is true about default constructor?**

It's always public

It calls super() and initializes instance variables

It's provided only if no constructors are defined

It takes parameters based on instance variables

Correct Answer: C) It's provided only if no constructors are defined

Explanation: Default constructor is provided by compiler only when no constructors are explicitly defined in the class.

Java Constructor Examples XLIII

Question 24: Static Block Execution

```
1 class StaticInit
2 {
3     static String s = "Start";
4     static
5     {
6         s = "Middle";
7     }
8     StaticInit()
9     {
10         System.out.print(s);
11     }
12     static
13     {
14         s = "End";
15     }
16     public static void main(String[] args)
17     {
18         new StaticInit();
19     }
20 }
```

What is the output?



Java Constructor Examples XLIV

Start

Middle

End

StartMiddleEnd

Correct Answer: C) End

Explanation: Static blocks execute in order before constructor, and the last assignment to static variable wins.

Java Constructor Examples XLV

Question 25: super() Placement Rule

```
1 class Base
2 {
3     Base()
4     {
5         System.out.print("Base ");
6     }
7 }
8 class Derived extends Base
9 {
10     Derived()
11     {
12         System.out.print("Derived ");
13     }
14     Derived(int x)
15     {
16         this();
17         super();
18         System.out.print("Derived" + x);
19     }
20 }
```

What happens?

Java Constructor Examples XLVI

Compiles successfully

Runtime error

Compilation error - `super()` must be first

Prints "Base Derived Derived10"

Correct Answer: C) Compilation error - `super()` must be first

Explanation: `super()` or `this()` must be the first statement in a constructor, not after `this()`.

Java Constructor Examples XLVII

Question 26: Final Variable Initialization

```
1 class FinalTest
2 {
3     final int x;
4     FinalTest()
5     {
6         System.out.print(x);
7         x = 10;
8         System.out.print(x);
9     }
10    public static void main(String[] args)
11    {
12        new FinalTest();
13    }
14 }
```

What is the output?

0 10

10 10

Java Constructor Examples XLVIII

Compilation error

Runtime error

Correct Answer: C) Compilation error

Explanation: Final variable 'x' must be initialized before use in constructor. Reading it before assignment causes error.

Java Constructor Examples XLIX

Question 27: Garbage Collection

```
1 class GC
2 {
3     GC()
4     {
5         System.out.print("Created ");
6     }
7     protected void finalize()
8     {
9         System.out.print("Destroyed ");
10    }
11    public static void main(String[] args)
12    {
13        new GC();
14        new GC();
15        System.gc();
16    }
17 }
```

How many objects eligible for GC?

0

Java Constructor Examples L

1

2

Cannot determine

Correct Answer: D) Cannot determine

Explanation: GC is not guaranteed to run immediately when `System.gc()` is called. JVM decides when to actually perform garbage collection.

Java Constructor Examples LI

Question 28: Private Constructor Inheritance

```
1 class PrivateBase
2 {
3     private PrivateBase()
4     {
5
6     }
7 }
8 class PrivateDerived extends PrivateBase
9 {
10     PrivateDerived()
11     {
12         super();
13     }
14 }
```

What is the output?

Compiles successfully

Runtime error

Java Constructor Examples LII

Compilation error - private constructor inaccessible

Creates object successfully

Correct Answer: C) Compilation error - private constructor inaccessible

Explanation: Private constructors are not accessible to subclasses, preventing inheritance.

Java Constructor Examples LIII

Question 29: Early Return in Constructor

```
1 class ReturnTest
2 {
3     ReturnTest()
4     {
5         return;
6     }
7     ReturnTest(int x)
8     {
9         if(x < 0) return;
10        System.out.print("Positive");
11    }
12    public static void main(String[] args)
13    {
14        new ReturnTest(-1);
15    }
16 }
```

What happens?

Positive

Java Constructor Examples LIV

No output

Compilation error

Runtime error

Correct Answer: B) No output

Explanation: Return statement in constructor exits early without error, skipping the print statement.

Java Constructor Examples LV

Question 30: Factory Method Initialization

```
1 class Mixed
2 {
3     Mixed()
4     {
5         System.out.print("Constructor ");
6     }
7     static Mixed getMixed()
8     {
9         return new Mixed();
10    }
11    {
12        System.out.print("Initializer ");
13    }
14    public static void main(String[] args)
15    {
16        Mixed m = getMixed();
17    }
18 }
```

What is the output?

Java Constructor Examples LVI

Constructor Initializer

Initializer Constructor

Constructor

Initializer

Correct Answer: B) Initializer Constructor

Explanation: Instance initializer runs before constructor body, regardless of how object is created (factory method or direct).