1. **Given the following code snippet, which option correctly explains the use of `this()`?**

```
public class Example {
    int x;
    public Example() {
        this(10);
    }
    public Example(int x) {
        this.x = x;
    }
}
```

A) It calls a static method in the same class.
B) It invokes another constructor in the same class to initialize the field.
C) It calls the superclass constructor.
D) It creates a new object instance.

2. **Examine the code below. What does the `super()` call in the subclass constructor ensure?**

```
class Parent {
    Parent() {
        System.out.println("Parent Constructor");
    }
}
class Child extends Parent {
    Child() {
        super();
        System.out.println("Child Constructor");
    }
}
```

A) It initializes the Child class's own fields.
B) It invokes a method of the Parent class.
C) It calls the Parent class's no-argument constructor before executing the Child's constructor body.
D) It delays initialization until later in the Child constructor.

3. **Consider the following multi-level constructor chaining example. What is the correct order of constructor calls?**

```
class A {
    A() { System.out.println("A"); }
}
class B extends A {
    B() {
        super();
        System.out.println("B");
    }
}
class C extends B {
    C() {
```

```
        super();
        System.out.println("C");
    }
}
```

A) A, C, B
B) B, A, C
C) A, B, C
D) C, B, A

4. **Identify the error in the following code snippet regarding constructor chaining:**

```
public class ErrorExample {
    public ErrorExample() {
        System.out.println("Default Constructor");
        this(5);
    }
    public ErrorExample(int x) {
        System.out.println("Parameterized Constructor: " + x);
    }
}
```

A) `this(5)` should be replaced with `super(5)`. B) The call to `this(5)` must be the first statement in the constructor.
C) There is no error; the code is correct.
D) The constructor should not print any statements.

5. **Review the following code and select the option that best describes its output when creating a new instance of `Demo`:**

```
public class Demo {
    public Demo() {
        this(100);
        System.out.println("Default Constructor");
    }
    public Demo(int num) {
        System.out.println("Parameterized Constructor: " + num);
    }
    public static void main(String[] args) {
        new Demo();
    }
}
```

A) Only "Parameterized Constructor: 100" is printed.
B) "Default Constructor" is printed before "Parameterized Constructor: 100".
C) "Parameterized Constructor: 100" is printed first, then "Default Constructor".
D) The program throws a runtime exception.

6. **What is the output of the following code snippet?**

```
class Base {
    int value = 5;
```

```
}
class Derived extends Base {
    int value = 10;
    void printValues() {
        System.out.println(value);
        System.out.println(super.value);
    }
    public static void main(String[] args) {
        new Derived().printValues();
    }
}
```

A) 10 and 5
B) 5 and 10
C) 10 and 10
D) 5 and 5

7. **In the code snippet below, which statement best describes the use of `super.fieldName`?**

```
class Animal {
    String type = "Animal";
}
class Dog extends Animal {
    String type = "Dog";
    void displayType() {
        System.out.println(type);
        System.out.println(super.type);
    }
}
```

A) It accesses the Dog class's own type field.
B) It accesses the Animal class's type field, bypassing the subclass's field.
C) It causes a compile-time error because of field hiding.
D) It initializes both fields to the same value.

8. **What happens when a subclass does not call `super()` explicitly, given the following classes?**

```
class Parent {
    Parent() {
        System.out.println("Parent");
    }
}
class Child extends Parent {
    Child() {
        System.out.println("Child");
    }
}
```

A) The code fails to compile because `super()` is missing.
B) The compiler automatically inserts a call to the no-argument constructor of `Parent`. C) The Child constructor never executes.
D) The program throws a runtime exception.

9. **Analyze the following code. What is the main difference between `this()` and `super()` as used here?**

```
class Person {
    String name;
    Person(String name) { this.name = name; }
}
class Employee extends Person {
    int id;
    Employee(String name, int id) {
        super(name);
        this.id = id;
    }
}
```

A) `this()` would have been used to call the Person constructor, but `super()` is used instead.
B) `super()` is used to call the superclass constructor while `this()` is used for calling another constructor in the same class.
C) Both `this()` and `super()` perform the same function.
D) `this()` is only used for methods, not constructors.

10. **Examine the following code. What is the role of `super()` in the `Manager` constructor?**

```
class Employee {
    Employee() {
        System.out.println("Employee initialized");
    }
}
class Manager extends Employee {
    Manager() {
        super();
        System.out.println("Manager initialized");
    }
}
```

A) It initializes Manager-specific fields.
B) It delays the execution of the Manager constructor.
C) It ensures that the Employee constructor is executed before the Manager constructor body.
D) It creates a new Employee instance independent of Manager.

11. **Given the code below, what does constructor chaining help achieve?**

```
public class Config {
    int mode;
    String setting;
    public Config() {
        this(1, "Default");
    }
    public Config(int mode, String setting) {
        this.mode = mode;
```

```
        this.setting = setting;
    }
}
```

A) It makes the program run faster by eliminating constructors.
B) It reduces code duplication by having one constructor delegate initialization to another.
C) It allows the creation of multiple objects simultaneously.
D) It avoids the use of the `super()` call entirely.

12. **Review the following code. Which option best explains the risk of improper constructor chaining that might lead to recursion?**

```
public class Loop {
    public Loop() {
        this();
    }
}
```

A) It causes a compile-time error due to recursion in constructor chaining.
B) It causes a runtime error by infinite recursion, eventually leading to a stack overflow.
C) It is valid code and runs normally.
D) It calls the superclass constructor repeatedly.

13. **Which of the following best describes the concept and benefits of constructor chaining in Java?**

A) It allows multiple constructors to execute independently without sharing code.
B) It enables one constructor to call another, thereby centralizing common initialization and reducing redundancy.
C) It forces all constructors to call the same static method.
D) It is used to override the behavior of the superclass constructor.

14. **Consider the following multi-level inheritance code. Which option correctly states the order of execution for the constructors?**

```
class Grandparent {
    Grandparent() { System.out.print("G"); }
}
class Parent extends Grandparent {
    Parent() { System.out.print("P"); }
}
class Child extends Parent {
    Child() { System.out.print("C"); }
}
public class Test {
    public static void main(String[] args) {
        new Child();
    }
}
```

A) C, P, G
B) G, P, C
C) P, G, C
D) G, C, P

15. **Which potential pitfall is most likely when misusing `super()` in a subclass constructor?**

    A) It may cause the subclass to bypass its own initialization code.
    B) It might lead to a compile-time error if not placed as the first statement.
    C) It results in an automatic call to `this()` instead of the superclass constructor.
    D) It will initialize static variables incorrectly.

Scenario questions

Question 1: Accessing Parent and Child Class Variables Using `this` and `super`

You are designing a **Person-Student** system where:

- **Person** class has a `name` variable initialized in its constructor.
- **Student** class extends **Person** and also has a `name` variable.

Task:

- Create a `Student` class that **hides** the parent class variable `name` by defining a variable with the same name.
- Create a method in `Student` to print **both the parent's `name` and child's `name`** using `this` and `super`.

Expected Output Example:

```
Parent Name: John Doe
Child Name: Alice Smith
```

Question 2: Calling Parent Class Method Using `super`

You are developing a **Vehicle-Car** system where:

- **Vehicle** has a method `describe()` that prints `"This is a vehicle"`.
- **Car** extends **Vehicle** and **overrides** `describe()` to print `"This is a car"`.

Task:

- Inside `Car`, **override** the `describe()` method.
- Call the **parent class's `describe()` method** from within the `Car` class using `super`.

Expected Output Example:

```
This is a car
This is a vehicle
```

Question 3: Using `this()` for Constructor Chaining

You are creating a **Book-EBook** system where:

- **Book** has:
  - A **default constructor** that prints `"Default Book Constructor"`.
  - A **parameterized constructor** that takes a `title` and prints `"Book Title: <title>"`.
- **EBook** extends **Book**.

Task:

- In `EBook`, create a **default constructor** that **calls its own parameterized constructor** using `this()`.
- The parameterized constructor of `EBook` should print `"EBook Title: <title>"`.

Expected Output Example:

```
EBook Title: Java Programming
```

Question 4: Using `super()` to Call Parent Constructor

You are implementing an **Employee-Manager** system where:

- **Employee** has a constructor that prints `"Employee Created"`.
- **Manager** extends **Employee** and has its own constructor that prints `"Manager Created"`.

Task:

- Modify the `Manager` constructor to **first call** the `Employee` constructor using `super()`.
- If `super()` is removed, what happens? Test and observe.

Expected Output Example:

```
Employee Created
Manager Created
```
