1. Variable Shadowing

a. Instance Variable Shadowing

Occurs when a local variable or method parameter has the same name as an instance variable.

Java

```java
public class InstanceShadowing {
    private int x = 10; // Instance variable

    public void method(int x) { // Method parameter shadows instance variable
        System.out.println("Method parameter x: " + x); // Refers to method parameter
        System.out.println("Instance variable x: " + this.x); // Refers to instance variable
    }

    public static void main(String[] args) {
         InstanceShadowing obj = new InstanceShadowing();
         obj.method(20);
     }
}
```

b. Class Variable (Static Variable) Shadowing

Occurs when a local variable or method parameter has the same name as a static variable.

Java

```java
public class StaticShadowing {
    private static int x = 10; // Static variable

    public static void staticMethod() {
        int x = 20; // Local variable shadows static variable
        System.out.println("Local variable x: " + x); // Refers to local variable
        System.out.println("Static variable x: " + StaticShadowing.x); // Refers to static variable
    }

     public static void main(String[] args) {
         StaticShadowing.staticMethod();
     }
}
```

c. Method Parameter Shadowing

Occurs when a local variable within a method has the same name as a method parameter.

Java

```java
public class ParameterShadowing {
    public void method(int x) {
        System.out.println("Method parameter x: " + x); // Refers to method parameter
        {
            int x = 30; // Local variable shadows method parameter
            System.out.println("Local variable x: " + x); // Refers to local variable
        }
        System.out.println("Method parameter x after block: " + x); // Refers to method parameter
    }

     public static void main(String[] args) {
         ParameterShadowing obj = new ParameterShadowing();
         obj.method(20);
     }
}
```

d. Inheritance and Variable Shadowing

Occurs when a subclass declares a variable with the same name as a variable in its superclass.

Java

 Collapse

```java
class SuperClass {
    int x = 10;
}
```

```java

class SubClass extends SuperClass {
    int x = 20; // Shadows superclass variable

    public void display() {
        System.out.println("SubClass x: " + x); // Refers to subclass variable
        System.out.println("SuperClass x: " + super.x); // Refers to superclass variable
    }
}

public class InheritanceShadowing {
    public static void main(String[] args) {
        SubClass obj = new SubClass();
        obj.display();
    }
}
```

2. Method Shadowing (Static Method Hiding)

Occurs when a subclass defines a static method with the same signature as a static method in its superclass. This is known as method hiding, not overriding.

Java

Collapse

```java
class SuperClass {
    static void staticMethod() {
        System.out.println("SuperClass static method");
    }
}

class SubClass extends SuperClass {
    static void staticMethod() { // This hides the superclass method
        System.out.println("SubClass static method");
    }
}

public class StaticMethodShadowing {
    public static void main(String[] args) {
        SuperClass superClass = new SuperClass();
        SubClass subClass = new SubClass();
        SuperClass ref = new SubClass();

        superClass.staticMethod(); // Calls SuperClass's static method
        subClass.staticMethod();   // Calls SubClass's static method
        ref.staticMethod();        // Calls SuperClass's static method due to reference type
    }
}
```

Key Points

Scope Precedence: The most immediate scope takes precedence. Local variables shadow method parameters, which in turn shadow instance and static variables.

Accessing Shadowed Variables: Use this to access shadowed instance variables and super to access shadowed superclass variables. Use ClassName.variableName to access shadowed static variables.

Static Method Hiding: Static methods are hidden, not overridden. The method call is determined by the reference type, not the runtime type of the object.

Readability: While shadowing is allowed, it can lead to confusion and make code harder to read. It's generally a good practice to avoid shadowing by using distinct variable and method names.

Understanding these shadowing concepts helps in writing clear and maintainable code, avoiding potential bugs related to variable and method scope and access.