L41

Friday, February 4, 2022 1:02 AM

Local Classes, Anonymous Inner Classses

Local classes

Q1. A nested class which is declared inside a block (between an opening and closing brace) is called a local class.

Inner classes are also local classes. However, to differentiate between inner and local classes, we use the term local class to a class which is declared inside a method or a conditional or a loop block. For example:

```
class A { //this is called a top-level classclass B { //this is an inner class}
        static class C { //this is an inner class}
        public void someMethodInClassA() {
                class D {
                }
        }
}
```

In the above code,

1. A is called a **top-level class**
2. B is called an **inner class**
3. C is called a **static nested class** and
4. D is called a **local class**

Local classes can access instance members of their enclosing class.

The visibility of the local class is restricted to the scope of the enclosing block (braces).

Note that local classes **cannot have static** methods or interfaces as members. We can have fields marked as static provided they are also marked as final (such fields are called as constants).

See and retype the below code.

```java
package q11296;
public class A {
    private int value = 7;
    class B {
        private B() {
            System.out.println("In inner class B's constructor");
        }
    }
    static class C {
        private C() {
            System.out.println("In static nested class C's constructor");
        }
    }
    public void someMethodInClassA() {
        class D {
            private D() {
                System.out.println("In local class D's constructor");
                System.out.println("value = " + value);
            }
        }
        D d = new D();
    }
    public static void main(String[] args) {
        A a = new A();
        A.B b = new A().new B();
        A.C c = new A.C();
        a.someMethodInClassA();
    }
}

```

Anonymous inner classses

Q1. An anonymous class is very similar to a local class except that an anonymous class combines **both the declaration and definition** of the class into a single expression statement.

Secondly an anonymous class usually implements an interface or extends a class.

For example the below code shows the **difference between** an **inner class** and an **anonymous class**:

```
interface Printer { //this is a top-level interfacepublic void printMe();
}
class A {  //this is a normal top-level classpublic static void main(String[] args) {
                class PrinterImpl implements Printer { // an example of a normal local classpublic void printMe() {
                                //do something....
                        }
                }
Printer myPrinter1 = new PrinterImpl();
Printer myPrinter2 = new Printer() {// an example of an anonymous class public void printMe() {
                                //do something....
                        }
                };

                myPrinter1.printMe();
                myPrinter2.printMe();
        }
}
```

In the above code, PrinterImpl is a **local class** that implements the interface Printer. An instance of this local class is created using the statement:

```
Printer myPrinter1 = new PrinterImpl();
```

The **anonymous class** which is initializing the reference myPrinter2 has the below properties:

1. Anonymous classes do not have names
2. An anonymous class is an expression containing a block of code which is terminated by a semicolon ;
3. Anonymous class instantiation is done using the new keyword, just like a constructor invocation
4. Anonymous class can either **implement** an interface and there by provide the implementation for its methods or **extend** a class and override required methods.
5. When an anonymous class is implementing an interface, the new keyword is prefixed to the interface name, that is followed by empty parenthesis.
6. When an anonymous class is extending a class, the new keyword is prefixed to one of the existing constructors in that class.

See and retype the below code.

```
package q11297;
interface Printer {
    public void printMe();
}
class A {
    public static void main(String[] args) {
        class PrinterImpl implements Printer {
            public void printMe() {
                System.out.println("I am in printMe method of the local class PrinterImpl instance");
            }
        }
        Printer myPrinter1 = new PrinterImpl();
        Printer myPrinter2 = new Printer() {
            public void printMe() {
                System.out.println("I am in printMe method of the anonymous class");
            }
        };
        myPrinter1.printMe();
        myPrinter2.printMe();
    }
}
```

Q2. See and retype the below code. It has two examples of anonymous classes. One is created by implementing an **interface** Printer and the other by extending a **class** named Prefixer.

Note that when an anonymous class is implementing an interface, the new keyword is prefixed to the interface name, which is followed by empty parenthesis.

And note that when an anonymous class is extending a **class**, the new keyword is prefixed to one of the existing constructors in that class.

```
package q11298;
interface Printer {
    public void printMe();
}
class Prefixer {
    protected String prefix;
    public Prefixer(String prefix) {
        this.prefix = prefix;
    }
    public String getPrefixedName(String name) {
        return prefix + " " + name;
    }
}
class AnonymousExample {
    public static void main(String[] args) {
        Printer printer = new Printer() {
            public void printMe() {
                System.out.println("printMe is called!");
            }
        };
        Prefixer doublePrefixer = new Prefixer("Hello") {
            public String getPrefixedName(String name) {
                return prefix + " " + prefix + " " + name;
            }
        };
        printer.printMe();
        System.out.println(doublePrefixer.getPrefixedName("James"));
    }
}
```

Q3. An anonymous class **can**:

1. have its own fields and custom methods which may not be there in the interface it is implementing or the class it is extending
2. access members of its enclosing class
3. have static fields provided they are also marked as final
4. have local classes

An anonymous class **cannot**:

1. be reused. They are just like expressions, whose complete implementation is provided in the accompanying code block.
2. access non-final local variables that are declared in its enclosing block
3. have static methods or interfaces

Select all the correct statements from the below code:

```
interface Printer {
	public void printMe();
}
class Prefixer {
	protected String prefix;
	public Prefixer(String prefix) {
		this.prefix = prefix;
	}
	public String getPrefixedName(String name) {
		return prefix + " " + name;
	}
}
class AnonymousExample {
	public static void main(String[] args) {
		Printer printer = new Printer(String text) { // statement 1this.text = text; // statement 2public void printMe() {
				System.out.println("printMe is called!");
			}
			public static void printMe2() { // statement 3System.out.println("printMe2 is called!");
```

```
                }
        };
        Prefixer doublePrefixer = new Prefixer("Hello") { // statement 4public String getOnlyPrefix() { // statement 5return prefix;
                }
                public String getPrefixedName(String name) {
                        return prefix + " " + prefix + " " + name;
                }
        };
        printer.printMe();
        System.out.println(doublePrefixer.getPrefixedName("James"));
    }
}
```

- [x] Statements 1 and 2 will result in compilation errors.
- [ ] Statement 3 will not result in a compilation error, since anonymous classes can have custom methods.
- [ ] Statement 4 will result in a compilation error as it is taking an argument `"Hello"`, while there should have been only empty parenthesis.
- [ ] Statement 5 will result in compilation error, because it cannot declare extra methods other than those declared in the class `Prefixer`.