Lecture 06: A Closer Look at Methods and Classes

Mahesh Kumar
Assistant Professor (Adhoc)

Department of Computer Science
Acharya Narendra Dev College
University of Delhi

Course webpage
[http://www.mkbhandari.com/mkwiki]

Outline

1. Overloading Methods
2. Overloading Constructors
3. Using Objects as Parameters
4. A Closer Look at Argument Passing
5. Returning Objects

Overloading Methods

- In Java it is possible to define two or more methods within the same class that share the same name, as long as their parameter declarations are different.
- Method overloading is one of the ways that Java supports polymorphism.
- When an overloaded method is invoked, Java uses the type and/or number of arguments as its guide to determine which version of the overloaded method to actually call.
- While overloaded methods may have different return types, the return type alone is insufficient to distinguish two versions of a method.
- When Java encounters a call to an overloaded method, it simply executes the version of the method whose parameters match the arguments used in the call.

Overloading Methods

```
// Method overloading Demo
class OverloadDemo {
        void test() {
                System.out.println("No parameters");
        }

        // Overload test for one integer parameter.
        void test(int a) {
                System.out.println("a: " + a);
        }

        // Overload test for two integer parameters.
        void test(int a, int b) {
                System.out.println("a and b: " + a + " " + b);
        }

        // Overload test for a double parameter
        double test(double a) {
                System.out.println("double a: " + a);
                return a*a;
        }
}
```

Overloading Methods

```
// Method overloading Demo
class OverloadDemo {
        void test() {
                System.out.println("No parameters");
        }

        // Overload test for one integer parameter.
        void test(int a) {
                System.out.println("a: " + a);
        }

        // Overload test for two integer parameters.
        void test(int a, int b) {
                System.out.println("a and b: " + a + " " + b);
        }

        // Overload test for a double parameter
        double test(double a) {
                System.out.println("double a: " + a);
                return a*a;
        }
}
```

```
class Overload {
        public static void main(String args[ ]) {
                OverloadDemo ob = new OverloadDemo( );
                double result;
                // call all versions of test( ) - overloaded 4 times
                ob.test( );
                ob.test(10);
                ob.test(10, 20);
                result = ob.test(123.25);
                System.out.println("Result of ob.test(123.25): " + result);
        }
}
```

Overloading Methods

```
// Method overloading Demo
class OverloadDemo {
        void test() {
                System.out.println("No parameters");
        }

        // Overload test for one integer parameter.
        void test(int a) {
                System.out.println("a: " + a);
        }

        // Overload test for two integer parameters.
        void test(int a, int b) {
                System.out.println("a and b: " + a + " " + b);
        }

        // Overload test for a double parameter
        double test(double a) {
                System.out.println("double a: " + a);
                return a*a;
        }
}
```

```
class Overload {
        public static void main(String args[ ]) {
                OverloadDemo ob = new OverloadDemo( );
                double result;
                // call all versions of test( ) - overloaded 4 times
                ob.test( );
                ob.test(10);
                ob.test(10, 20);
                result = ob.test(123.25);
                System.out.println("Result of ob.test(123.25): " + result);
        }
}
```

Q What will be the output?

Overloading Methods

```
// Method overloading Demo
class OverloadDemo {
        void test() {
                System.out.println("No parameters");
        }

        // Overload test for one integer parameter.
        void test(int a) {
                System.out.println("a: " + a);
        }

        // Overload test for two integer parameters.
        void test(int a, int b) {
                System.out.println("a and b: " + a + " " + b);
        }

        // Overload test for a double parameter
        double test(double a) {
                System.out.println("double a: " + a);
                return a*a;
        }
}
```

```
class Overload {
        public static void main(String args[ ]) {
                OverloadDemo ob = new OverloadDemo( );
                double result;
                // call all versions of test( ) - overloaded 4 times
                ob.test( );
                ob.test(10);
                ob.test(10, 20);
                result = ob.test(123.25);
                System.out.println("Result of ob.test(123.25): " + result);
        }
}
```

Q What will be the output?

```
No parameters
a: 10
a and b: 10 20
double a: 123.25
Result of ob.test(123.25): 15190.5625
```

Overloading Methods – Automatic Type Conversions

- When an overloaded method is called, Java looks for a match between the arguments used to call the method and the method's parameters.
- However, this match need not always be exact.
- In some cases, Java's automatic type conversions can play a role in overload resolution.

Byte –> Short –> Int –> Long – > Float –> Double

Widening or Automatic Conversion

Double –> Float –> Long –> Int –> Short –> Byte

Narrowing or Explicit Conversion

Overloading Methods – Automatic Type Conversions


```
// Automatic type conversions apply to overloading.

class OverloadATDemo {
        void test( ) {
                System.out.println("No parameters");
        }

        // Overload test for two integer parameters.
        void test(int a, int b) {
                System.out.println("a and b: " + a + " " + b);
        }

        // Overload test for a double parameter
        void test(double a) {
                System.out.println("Inside test(double) a: " + a);
        }
}
```

Overloading Methods – Automatic Type Conversions

```
// Automatic type conversions apply to overloading.

class OverloadATDemo {
        void test( ) {
                System.out.println("No parameters");
        }

        // Overload test for two integer parameters.
        void test(int a, int b) {
                System.out.println("a and b: " + a + " " + b);
        }

        // Overload test for a double parameter
        void test(double a) {
                System.out.println("Inside test(double) a: " + a);
        }
}
```

```
class Overload {
        public static void main(String args[ ]) {
                OverloadATDemo ob = new OverloadATDemo( );
                int i = 88;
                ob.test( );

                // this will invoke test(int,int)
                ob.test(10, 20);

                // this will invoke test(double)
                ob.test(i);

                // this will invoke test(double)
                ob.test(123.2);
        }
}
```

Overloading Methods – Automatic Type Conversions

```
// Automatic type conversions apply to overloading.

class OverloadATDemo {
        void test( ) {
                System.out.println("No parameters");
        }

        // Overload test for two integer parameters.
        void test(int a, int b) {
                System.out.println("a and b: " + a + " " + b);
        }

        // Overload test for a double parameter
        void test(double a) {
                System.out.println("Inside test(double) a: " + a);
        }
}
```

```
class Overload {
        public static void main(String args[ ]) {
                OverloadATDemo ob = new OverloadATDemo( );
                int i = 88;
                ob.test( );

                // this will invoke test(int,int)
                ob.test(10, 20);

                // this will invoke test(double)
                ob.test(i);

                // this will invoke test(double)
                ob.test(123.2);
        }
}
```

Q What will be the output?

Overloading Methods – Automatic Type Conversions

```
// Automatic type conversions apply to overloading.

class OverloadATDemo {
        void test( ) {
                System.out.println("No parameters");
        }

        // Overload test for two integer parameters.
        void test(int a, int b) {
                System.out.println("a and b: " + a + " " + b);
        }

        // Overload test for a double parameter
        void test(double a) {
                System.out.println("Inside test(double) a: " + a);
        }
}
```

```
class Overload {
        public static void main(String args[ ]) {
                OverloadATDemo ob = new OverloadATDemo( );
                int i = 88;
                ob.test( );

                // this will invoke test(int,int)
                ob.test(10, 20);

                // this will invoke test(double)
                ob.test(i);

                // this will invoke test(double)
                ob.test(123.2);
        }
}
```

Q What will be the output?

No parameters
a and b: 10 20
Inside test(double) a: 88
Inside test(double) a: 123.2

Overloading Constructors

```
// Here, Box defines three constructors to initialize the dimensions of a box various ways.
class Box {
        double width;  double height; double depth;
        // constructor used when all dimensions specified
        Box(double w, double h, double d) {
                width = w;
                height = h;
                depth = d;
        }
        // constructor used when no dimensions specified
        Box( ) {
                width = -1;
                height = -1;
                depth = -1;
        }
        // constructor used when cube is created
        Box(double len) {
                width = height = depth = len;
        }
        // compute and return volume
        double volume( ) {
                return width * height * depth;
        }
}
```

```
class OverloadCons {
        public static void main(String args[ ]) {
                // create boxes using the various constructors
                Box mybox1 = new Box(10, 20, 15);
                Box mybox2 = new Box( );
                Box mycube = new Box(7);
                double vol;
                // get volume of first box
                vol = mybox1.volume( );
                System.out.println("Volume of mybox1 is " + vol);
                // get volume of second box
                vol = mybox2.volume( );
                System.out.println("Volume of mybox2 is " + vol);
                // get volume of cube
                vol = mycube.volume( );
                System.out.println("Volume of mycube is " + vol);
        }
}
```

Q What will be the output?

Using Objects as Parameters

```
// Objects may be passed to methods.
class Test {
        int a, b;
        Test(int i, int j) {
                a = i;
                b = j;
        }
        // return true if o is equal to the invoking object
        boolean equalTo(Test o) {
                if(o.a == a && o.b == b)
                        return true;
                else
                        return false;
        }
}
```

```
class PassOb {
        public static void main(String args[ ]) {
                Test ob1 = new Test(100, 22);
                Test ob2 = new Test(100, 22);
                Test ob3 = new Test(-1, -1);
                System.out.println("ob1 == ob2: " + ob1.equalTo(ob2));
                System.out.println("ob1 == ob3: " + ob1.equalTo(ob3));
        }
}
```

Q What will be the output?

ob1 == ob2: true
ob1 == ob3: false

Copy Constructor

```
// Here, Box allows one object to initialize another.
class Box {
        double width;
        double height;
        double depth;
        // Notice this constructor. It takes an object of type Box.
        Box(Box ob) { // pass object to constructor
                width = ob.width;
                height = ob.height;
                depth = ob.depth;
        }
        // constructor used when all dimensions specified
        Box(double w, double h, double d) {
                width = w;
                height = h;
                depth = d;
        }
        // constructor used when no dimensions specified
        Box( ) {
                width = -1;
                height = -1;
                depth = -1;
        }
        // constructor used when cube is created
        Box(double len) {
                width = height = depth = len;
        }
        // compute and return volume
        double volume( ) {
                return width * height * depth;
        }
}
```

Copy Constructor

```
class OverloadCons2 {
        public static void main(String args[ ]) {
                // create boxes using the various constructors
                Box mybox1 = new Box(10, 20, 15);
                Box mybox2 = new Box( );
                Box mycube = new Box(7);

                // create copy of mybox1
                Box myclone = new Box(mybox1);

                double vol;
                // get volume of first box
                vol = mybox1.volume();
                System.out.println("Volume of mybox1 is " + vol);
                // get volume of second box
                vol = mybox2.volume();
                System.out.println("Volume of mybox2 is " + vol);
                // get volume of cube
                vol = mycube.volume();
                System.out.println("Volume of cube is " + vol);
                // get volume of cube
                vol = mycube.volume( );
                System.out.println("Volume of cube is " + vol);
                // get volume of clone
                vol = myclone.volume( );
                System.out.println("Volume of clone is " + vol);
        }
}
```

Q What will be the output?

A Closer Look at Argument Passing

- In general, there are two ways that a computer language can pass an argument to a subroutine.

1. *Call-by-value*
   - Copies the value of an argument into the formal parameter of the subroutine.
   - Changes made to the parameter of the subroutine have no effect on the argument.

2. *Call-by-Reference*
   - A reference to an argument (not the value of the argument) is passed to the parameter.
   - Inside the subroutine, this reference is used to access the actual argument specified in the call.
   - Changes made to the parameter will affect the argument used to call the subroutine.

A Closer Look at Argument Passing


```
// Primitive types are passed by value.
class Test {
        void meth(int i, int j) {
                i *= 2;
                j /= 2;
        }
}

class CallByValue {
        public static void main(String args[ ]) {
                Test ob = new Test();
                int a = 15, b = 20;
                System.out.println("a and b before call: " + a + " " + b);
                ob.meth(a, b);
                System.out.println("a and b after call: " +a + " " + b);
        }
}
```

A Closer Look at Argument Passing

```
// Primitive types are passed by value.
class Test {
        void meth(int i, int j) {
                i *= 2;
                j /= 2;
        }
}

class CallByValue {
        public static void main(String args[ ]) {
                Test ob = new Test();
                int a = 15, b = 20;
                System.out.println("a and b before call: " + a + " " + b);
                ob.meth(a, b);
                System.out.println("a and b after call: " +a + " " + b);
        }
}
```

Q What will be the output?

a and b before call: 15 20
a and b after call: 15 20

A Closer Look at Argument Passing

```
// Objects are passed through their references.
class Test {
        int a, b;
        Test(int i, int j) {
                a = i;
                b = j;
        }
        // pass an object
        void meth(Test o) {
                o.a *= 2;
                o.b /= 2;
        }
}
class PassObjRef {
        public static void main(String args[ ]) {
                Test ob = new Test(15, 20);
                System.out.println("ob.a and ob.b before call: " + ob.a + " " + ob.b);
                ob.meth(ob);
                System.out.println("ob.a and ob.b after call: " +ob.a + " " + ob.b);
        }
}
```

A Closer Look at Argument Passing

```
// Objects are passed through their references.
class Test {
        int a, b;
        Test(int i, int j) {
                a = i;
                b = j;
        }
        // pass an object
        void meth(Test o) {
                o.a *= 2;
                o.b /= 2;
        }
}
class PassObjRef {
        public static void main(String args[ ]) {
                Test ob = new Test(15, 20);
                System.out.println("ob.a and ob.b before call: " + ob.a + " " + ob.b);
                ob.meth(ob);
                System.out.println("ob.a and ob.b after call: " +ob.a + " " + ob.b);
        }
}
```

Q What will be the output?

ob.a and ob.b before call: 15 20
ob.a and ob.b after call: 30 10

Returning Objects

```
// Returning an object.
class Test {
        int a;
        Test(int i) {
                a = i;
        }
        Test incrByTen( ) {
                Test temp = new Test(a+10);
                return temp;
        }
}
```

```
class RetOb {
        public static void main(String args[ ]) {
                Test ob1 = new Test(2);
                Test ob2;
                ob2 = ob1.incrByTen( );
                System.out.println("ob1.a: " + ob1.a);
                System.out.println("ob2.a: " + ob2.a);
                ob2 = ob2.incrByTen();
                System.out.println("ob2.a after second increase: "+ ob2.a);
        }
}
```

Q What will be the output?

ob1.a: 2
ob2.a: 12
ob2.a after second increase: 22

References

Reference for this topic

- [Book: Java: The Complete Reference, Ninth Edition: Herbert Schildt]
  https://www.amazon.in/Java-Complete-Reference-Herbert-Schildt/dp/0071808558

- [Web: GeeksforGeeks]
  https://www.geeksforgeeks.org/java/

- [Web: Java T Point tutorial]
  https://www.javatpoint.com/java-tutorial