# Object Oriented Programming

Chapter 7

Ayesha Khatun

Lecturer, Green University of Bangladesh

# A Closer Look at Methods and Classes

1. Overloading Methods
2. Overloading Constructors
3. Introducing final
4. Understanding static
5. A Closer Look at Argument Passing

# Overloading Methods

- **Method Overloading** is a feature that allows a class to have more than one **method** having the same name, if their argument lists are different.

1. Same method/constructor name
2. Same class
3. Different argument

```
Class A{
int x;
 A(int id){
System.out.println(“Account Create!”);
}
A(int id, int name){
System.out.println(“Account Create, thank you for your name!”);

}
A(){
System.out.println(“Account is not Create!”);

}
}
```

# Overloading Methods

```
class Adder{
static int add(int a, int b){return a+b;}
static double add(double a, double b){return a+b;}
}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}}
```

# Overloading Constructors-129P

```
class Box {
double width, height, depth;
Box(double w, double h, double d) {
width = w;
height = h;
depth = d;}
Box() {
width = -1; // use -1 to indicate
height = -1; // an uninitialized
depth = -1; // box}
Box(double len) {
width = height = depth = len;}
double volume() {
return width * height * depth;}}
```

# Introducing final

- A variable can be declared as **final. Doing so prevents its contents from being modified.**
- This means that you must initialize a **final variable when it is declared. For example:**
- final int FILE_NEW = 1;
- final int FILE_OPEN = 2;
- final int FILE_SAVE = 3;
- final int FILE_SAVEAS = 4;
- final int FILE_QUIT = 5;

# Understanding static

Methods declared as **static have several restrictions:**

- They can only call other **static methods.**
- They must only access **static data.**
- They cannot refer to **this or super in any way. (The keyword super relates to**

inheritance and is described in the next chapter.)

# A Closer Look at Argument Passing

- Call by value
- Call by reference

# Call 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);
- }}
- The output from this program is shown here:
- a and b before call: 15 20
- a and b after call: 15 20

# Call by reference

```
class Test {
int a, b;
Test(int i, int j) {
a = i;
b = j;}
void meth(Test o) {
o.a *= 2;
o.b /= 2;
}}
```

# Call by reference

```
class CallByRef {
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);
}
}
```

This program generates the following output:

ob.a and ob.b before call: 15 20

ob.a and ob.b after call: 30 10

# Thank You!