CS205 Object Oriented Programming in Java

Module 2 - **Core Java Fundamentals (Part 10)**

Prepared by

Renetha J.B.

AP

Dept.of CSE,

Lourdes Matha College of Science and Technology

Topics

- Core Java Fundamentals**:**
  - ✓Inheritance :
    - ✓ Calling Order of Constructors
    - ✓ Method Overriding
    - ✓The Object class

Calling Order of Constructors

- Constructors are called in the order of derivation, **from superclass to subclass**
- When subclass object is created**, it first calls superclass constructor** then only it calls subclass constructor.
- If **super( ) is not used to call superclass constructor,** then the **default constructor of each superclass will be executed** before executing subclass constructors.

```
class A
{
int i;

A()
{
 System.out.println("Constructor of superclass A");
}
}
class B extends A
{
int j;
B()
{
  System.out.println("Constructor of subclass B");
}
}
```

```
class Consorder
{

  public static void main(String args[])
  {
    B obb =new B();
  }

}
```

OUTPUT
Constructor of superclass A
Constructor of subclass B

```
class A
{
int i;
A()
{
System.out.println("Constructor of superclass A");
}
}
class B extends A
{
int j;
B()
{
System.out.println("Constructor of subclass B");
}
}
class C extends B
{
int j;
C()
{
System.out.println("Constructor of subclass C");
}
}
class Consorder{

public static void main(String args[])
{
C obc =new C();
}

}
```

OUTPUT

Constructor of superclass A
Constructor of subclass B
Constructor of subclass C

Calling Order of Constructors(contd.)

- Superclass has no knowledge of any subclass, any initialization it needs to perform is separate and it should be done as a prerequisite to initialize the subclass object.
- Therefore, **superclass constructors are executed** before executing subclass constructors, when we create subclass object.

Method Overriding

- In a class hierarchy, when a **method in a subclass** has the same name and type signature as a **method in its superclass**, then the *method in the **subclass** is said to **override the method in the superclass.***
- This is called METHOD OVERRIDING

Method Overriding(contd.)

- When an overridden method is called from within a subclass, it will always ***refer to the method defined by the subclass***.
  - The version of *the method defined by the superclass will be hidden.*

- // Method overriding.

```
class A {
int i, j;
A(int a, int b) {
i = a;
j = b;
}
void show() {
System.out.println(" i : " + i + " j: " + j);
}
}
class B extends A {
int k;
B(int a, int b, int c) {
super(a, b);
k = c;
}
void show() {
System.out.println("k: " + k);
}
}
```

```
class Override {
public static void main(String args[])
{
B subOb = new B(1, 2, 3);
subOb.show(); // this calls show() in B
}
}
```

OUTPUT
k: 3

When **show( )** is invoked on an object of type B, the version of **show( ) defined within B is used**.
That is, the version of show( ) inside subclass B overrides the version declared in superclass A.

- // No method overriding.

```
class A {
int i, j;
A(int a, int b) {
i = a;
j = b;
}
void show() {
System.out.println(" i : " + i + " j: " + j);
}
}
class B extends A {
int k;
B(int a, int b, int c) {
super(a, b);
k = c;
}
void display() {
System.out.println("k: " + k);
}
}
```

```
class Sample{
public static void main(String args[])
{
B subOb = new B(1, 2, 3);
subOb.show(); // this calls show() in A
}
}
```

OUTPUT
i: 1 j: 2

Here when **show( )** is invoked on an object of type B, since the version of **show( ) is not defined within B** the version of show() declared in superclass A is called and excuted.

Method Overriding(contd.)

- To access the **superclass version of an overridden method**, we can do using **super** keyword**.**

```
• // Method overriding.
class A {
int i, j;
A(int a, int b) {
i = a;
j = b;
}
void show() {
System.out.println(" i : " + i + " j: " + j);
}
}
class B extends A {
int k;
B(int a, int b, int c) {
super(a, b);
k = c;
}
void show() {
super.show();
System.out.println("k: " + k);
}
}
```

```
class Override {
public static void main(String args[])
{
B subOb = new B(1, 2, 3);
subOb.show(); // this calls show() in B
}
}
```

OUTPUT
i:1 j:2
k: 3

When **show( )** is invoked on an object of type B, the version of **show( ) defined within B is used**..

super.show() calls the show() method in its superclas.

Method Overriding(contd.)

- **Method overriding** occurs only when the ***names and the type signatures of the methods*** in subclass and superclass are **identical**.
- If ***names and the type signatures of the two*** **methods are different**, then the two methods are simply **overloaded.**

```
class A {
int i, j;
A() {
i = 0;
j = 0;
}
void show() {
System.out.println(show in A);
}
}
class B extends A {
int k;
B() {
k = 0;
}
void show(String msg) {
System.out.println("show in subclass B");
}
}
```

```
class Sample {
public static void main(String args[]) {
B subOb = new B();
subOb.show("k is "); // this calls show() in B
subOb.show(); // this calls show() in A
}
```

show in subclass B
show in A

Here show() Methods have differing type signatures. So they are overloaded – **not overridden**

```
class A {
int i, j;
A(int a, int b) {
i = a;
j = b;
}
void show() {
System.out.println("i : " + i + "j: " + j);
}
}
class B extends A {
int k;
B(int a, int b, int c) {
super(a, b);
k = c;
}
void show(String msg) {
System.out.println(msg + k);
}
}
```

```
class Sample {
public static void main(String args[]) {
B subOb = new B(1, 2, 3);
subOb.show("k is ");
subOb.show();
}
```

Here show() Methods have differing type signatures. So they are overloaded **– not overridden**

Object

- There is one special class, **Object**, defined by Java.
- All other classes are subclasses of **Object.**
- That is, **Object is a superclass of all other classes.**
- Reference variable of type **Object** can refer to an object of any other class.

Methods in Object class

Method	Purpose
Object clone( )	Creates a new object that is the same as the object being cloned.
boolean equals(Object *object*)	Determines whether one object is equal to another.
void finalize( )	Called before an unused object is recycled.
Class getClass( )	Obtains the class of an object at run time.
int hashCode( )	Returns the hash code associated with the invoking object.
void notify( )	Resumes execution of a thread waiting on the invoking object.
void notifyAll( )	Resumes execution of all threads waiting on the invoking object.
String toString( )	Returns a string that describes the object.
void wait( ) void wait(long *milliseconds*) void wait(long *milliseconds*, int *nanoseconds*)	Waits on another thread of execution.

Methods in Object class(contd.)

- The methods **getClass( ), notify( ), notifyAll( ), and wait( )** are declared as final.
- The **equals( )** method **compares the** contents of two objects.
  - It returns **true if the objects are equivalent, and false otherwise.**
- The **toString( )** method **returns a string** that contains a description of the object on which it is called.
  - This method is automatically called when an object is output using **println( ).**
  - **Many classes override this method.**

Reference

- Herbert Schildt, Java: The Complete Reference, 8/e, Tata McGraw Hill, 2011.