Week_8 Notes:

Can we Overload or Override static methods in java ?

Let us first define Overloading and Overriding.

Overriding : Overriding is a feature of OOP languages like Java that is related to run-time polymorphism. A subclass (or derived class) provides a specific implementation of a method in superclass (or base class).
The implementation to be executed is decided at run-time and decision is made according to the object used for call. Note that signatures of both methods must be same.

Overloading: Overloading is also a feature of OOP languages like Java that is related to compile time (or static) polymorphism. This feature allows different methods to have same name, but different signatures, especially number of input parameters and type of input paramaters. Note that in both C++ and Java, methods cannot be overloaded according to return type.

Can we overload static methods?
The answer is 'Yes'. We can have two or more static methods with same name, but differences in input parameters. For example, consider the following Java program.

```
public class Test {
    public static void foo() {
        System.out.println("Test.foo() called ");
    }
    public static void foo(int a) {
        System.out.println("Test.foo(int) called ");
    }
    public static void main(String args[])
    {
        Test.foo();
        Test.foo(10);
    }
}
```

Output:

```
Test.foo() called

Test.foo(int) called
```

Can we overload methods that differ only by static keyword?
We cannot overload two methods in Java if they differ only by static keyword (number of parameters and types of parameters is same). See following Java program for example. This behaviour is same in C++

```
public class Test {
    public static void foo() {
        System.out.println("Test.foo() called ");
    }
    public void foo() { // Compiler Error: cannot redefine
foo()
```

```
        System.out.println("Test.foo(int) called ");
    }
    public static void main(String args[]) {
        Test.foo();
    }
}
```

Output: Compiler Error, cannot redefine foo()

Can we Override static methods in java?

We can declare static methods with same signature in subclass, but it is not considered overriding as there won't be any run-time polymorphism. Hence the answer is 'No'.

If a derived class defines a static method with same signature as a static method in base class, the method in the derived class hides the method in the base class.

```
/* Java program to show that if static method is redefined by
   a derived class, then it is not overriding. */

// Superclass
class Base {

    // Static method in base class which will be hidden in
subclass
    public static void display() {
        System.out.println("Static or class method from
Base");
    }

     // Non-static method which will be overridden in derived
class
     public void print()  {
         System.out.println("Non-static or Instance method
from Base");
    }
}

// Subclass
class Derived extends Base {

    // This method hides display() in Base
    public static void display() {
         System.out.println("Static or class method from
Derived");
    }

    // This method overrides print() in Base
    public void print() {
         System.out.println("Non-static or Instance method
from Derived");
   }
}
```

```
// Driver class
public class Test {
    public static void main(String args[ ])  {
       Base obj1 = new Derived();

       // As per overriding rules this should call to class
Derive's static
       // overridden method. Since static method can not be
overridden, it
       // calls Base's display()
       obj1.display();

       // Here overriding works and Derive's print() is
called
       obj1.print();
    }
}
```

Output:

```
Static or class method from Base

Non-static or Instance method from Derived
```

Following are some important points for method overriding and static methods in Java.

1) For class (or static) methods, the method according to the type of reference is called, not according to the object being referred, which means method call is decided at compile time.

2) For instance (or non-static) methods, the method is called according to the type of object being referred, not according to the type of reference, which means method calls is decided at run time.

3) An instance method cannot override a static method, and a static method cannot hide an instance method.

4) In a subclass (or Derived Class), we can overload the methods inherited from the superclass. Such overloaded methods neither hide nor override the superclass methods — they are new methods, unique to the subclass.

Accessing Grandparent's member in Java using super

Directly accessing Grandparent's member in Java:

Predict the output of following Java program.

```
// filename Main.java
class Grandparent {
    public void Print() {
        System.out.println("Grandparent's Print()");
    }
}

class Parent extends Grandparent {
    public void Print() {
        System.out.println("Parent's Print()");
    }
}
```

```
class Child extends Parent {
    public void Print() {
        super.super.Print();  // Trying to access
Grandparent's Print()
        System.out.println("Child's Print()");
    }
}

public class Main {
    public static void main(String[] args) {
        Child c = new Child();
        c.Print();
    }
}
```

Output: Compiler Error

There is error in line “super.super.print();”. In Java, a class cannot directly access the grandparent’s members. It is allowed in C++ though. In C++, we can use scope resolution operator (::) to access any ancestor’s member in inheritance hierarchy. In Java, we can access grandparent’s members only through the parent class. For example, the following program compiles and runs fine.

```
// filename Main.java
class Grandparent {
    public void Print() {
        System.out.println("Grandparent's Print()");
    }
}

class Parent extends Grandparent {
    public void Print() {
        super.Print();
        System.out.println("Parent's Print()");
    }
}

class Child extends Parent {
    public void Print() {
        super.Print();
        System.out.println("Child's Print()");
    }
}

public class Main {
    public static void main(String[] args) {
        Child c = new Child();
        c.Print();
    }
}
```

Output:

```
Grandparent's Print()

Parent's Print()

Child's Print()
```

Can we override private methods in Java?

Let us first consider the following Java program as a simple example of Overriding or Runtime Polymorphism.

```
class Base {
  public void fun() {
     System.out.println("Base fun");
  }
}

class Derived extends Base {
  public void fun() {  // overrides the Base's fun()
     System.out.println("Derived fun");
  }
  public static void main(String[] args) {
      Base obj = new Derived();
      obj.fun();
  }
}
```

The program prints “Derived fun”.
The Base class reference ‘obj’ refers to a derived class object (see expression “Base obj = new Derived()”). When fun() is called on obj, the call is made according to the type of referred object, not according to the reference.

Is Overiding possible with private methods?
Predict the output of following program.

```
class Base {
  private void fun() {
     System.out.println("Base fun");
  }
}

class Derived extends Base {
  private void fun() {
     System.out.println("Derived fun");
  }
  public static void main(String[] args) {
      Base obj = new Derived();
      obj.fun();
  }
}
```

We get compiler error “fun() has private access in Base” . So the compiler tries to call base class function, not derived class, means fun() is not overridden.