Method Overriding

Within a class hierarchy -

When a method in a subclass has the same return type and signature as a method in its superclass

then the method in the subclass overrides the method in the superclass.

```
class A {                          // Method overriding
     int i, j;
     A(int a, int b) {
          i = a;
          j = b;  }

     void show() {                 // display i and j
           System.out.println("i and j: " + i + " " + j);
     }
}
class B extends A {
    int k;

    B(int a, int b, int c) {
         super(a, b);
         k = c;
     }
                    // display k - this overrides show() in A
     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

The show( ) method in super class A did not execute otherwise values for i and j (1 and 2 in this case) would have been displayed.

An overridden method in the superclass can be accessed by using super as shown in this revised version of class B

```
class B extends A {             // revised version of class B
    int k;

    B(int a, int b, int c) {
        super(a, b);
        k = c;
    }
    void show() {
        super.show();          // this calls A's show()
        System.out.println("k: " + k);
    }
}
```

Output

```
i and j: 1  2
K: 3
```

Dynamic Method Dispatch

This is the mechanism by which a call to an overridden method is resolved at run time rather than compile time.

This is run-time polymorphism

When different types of objects are referred to, different versions of an overridden method will be used.

Dynamic Method Dispatch

run-time polymorphism

It is the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will execute.

```
// Demonstrate dynamic method dispatch.

class Suprclass {
  void who() {
    System.out.println("who() in Suprclass");
  }
}

class Sub1 extends Suprclass {
  void who() {
    System.out.println("who() in Sub1");
  }
}

class Sub2 extends Suprclass {
  void who() {
    System.out.println("who() in Sub2");
  }
}
```

```
class DynDispDemo {
    public static void main(String args[]) {
       Suprclass superOb = new Suprclass();
       Sub1 subOb1 = new Sub1();
       Sub2 subOb2 = new Sub2();

       Suprclass suprRef;

       suprRef = superOb;
       suprRef.who();

       suprRef = subOb1;
       suprRef.who();

       suprRef = subOb2;
       suprRef.who();
    }
}
```

OUTPUT

who() in Suprclass
who() in Sub1
who() in Sub2

The who version called is determined at runtime

Abstract Classes and methods

A method can be declared abstract if it doesn't contain code to implement its function.

The area method in the TwoDShape class doesn't calculate area. The subclass area methods do.

The super class method can be labeled abstract.

EX: abstract double area( );

A class containing an abstract method must also be labeled abstract.

No objects can be constructed from an abstract class.