Java

Inheritance

Inheritance

- Same inheritance concept of C++ in Java with some modifications
 - One class inherits the other using *extends* keyword
 - The classes involved in inheritance are known as superclass and subclass
 - *Multilevel* inheritance but no *multiple* inheritance
 - There is a special way to call the superclass's *constructor*
 - There is automatic *dynamic method dispatch*
- Inheritance provides code reusability (code of any class can be used by extending that class)

Simple Inheritance



Inheritance and Member Access

```
class M {
 1 🔍
              int i:
 2
              private int j;
 3
 4
              void set(int x, int y) {
 5
                   \mathbf{i} = \mathbf{x};
 6
 7
                   \mathbf{j} = \mathbf{v};
 8
 9
10
         class N extends M {
11
              int total;
12
13
              void sum() {
14
                   total = i + j;
15
                   // Error, j is not accessible here
16
17
        ≙}
18
10
```

```
20  public class SimpleInheritance2 {
21  public static void main(String[] args) {
22  N obj = new N();
23  obj.set(10, 20);
24  obj.sum();
25  System.out.println(obj.total);
26  }
27  }
```

- A class member that has been declared as private will remain private to its class
- It is not accessible by any code outside its class, including subclasses

Practical Example

```
class Box {
   3
            double width, height, depth;
 4
 5
            Box(Box ob) {
 6
                width = ob.width; height = ob.height; depth = ob.depth;
 7
            }
 8
 9
            Box(double w, double h, double d) {
10
                width = w; height = h; depth = d;
11
            }
12
13
            Box() { width = height = depth = 1; }
      +
14
17
            Box(double len) { width = height = depth = len; }
      +
18
21
            double volume() { return width * height * depth; }
22
      +
      ≙}
25
26
        class BoxWeight extends Box {
27
            double weight;
28
29
            BoxWeight(double w, double h, double d, double m) {
30
                width = w; height = h; depth = d; weight = m;
31
            }
32
      __}
33
```

Superclass variable reference to Subclass object



Using super to call Superclass Constructors

```
class BoxWeightNew extends Box {
 3
 4
          double weight;
                                           super() must always be the
 5
                                           first statement executed inside
 6
          BoxWeightNew(BoxWeightNew ob) {
 7
             super(ob);
                                           a subclass' constructor
 8
             weight = ob.weight;
9
10
11
          BoxWeightNew(double w, double h, double d, double m) {
12
             super(w, h, d);
13
             weight = m:
14
          }
15
16
          BoxWeightNew() {
17
             super(); // must be the 1st statement in constructor
18
             weight = 1;
19
20
21
          BoxWeightNew(double len, double m) {
22
             super(len);
23
             weight = m;
24
25
26
         void print() {
27
             System.out.println("Box(" + width + ", " + height +
28
                                  + depth + ", " + weight + ")");
29
                          Prepared By - Rifat Shahriyar
                                                                                     7
30
      }
```

Using super to call Superclass Constructors

```
31
32
      public class SuperTest {
33
          public static void main(String[] args) {
34
              BoxWeightNew box1 = new BoxWeightNew(10, 20, 15, 34.3);
35
              BoxWeightNew box2 = new BoxWeightNew(2, 3, 4, 0.076);
              BoxWeightNew box3 = new BoxWeightNew();
36
37
              BoxWeightNew cube = new BoxWeightNew(3, 2);
              BoxWeightNew clone = new BoxWeightNew(box1);
38
39
              box1.print();
40
              box2.print();
              box3.print();
41
42
              cube.print();
43
              clone.print();
44
45
46
      3
47
```

Using super to access Superclass hidden members

```
class C {
   3
            int i;
 4
            void show() {
 5
   6
 7
8
        Class D extends C {
9
            int i; // this i hides the i in C
10
11
            D(int a, int b) {
12
                 super.i = a; // i in C
13
                 i = b; // i in D
14
15
16
            void show() {
17 0
                 System.out.println("i in superclass: " + super.i);
18
                 System.out.println("i in subclass: " + i);
19
                 super.show();
20
21
22
23
        public class UseSuper {
24
            public static void main(String[] args) {
25
                 D \text{ subOb} = \mathbf{new} D(a; 1, b; 2);
26
                 subOb.show();
27
28
29
                           Prepared By - Rifat Shahriyar
20
```

Multilevel Inheritance

```
3 ෧↓
    int a;
 5
          XO {
 6
              System.out.println("Inside X's constructor");
 7
          3
 8
     |}
9
10 🔍
      class Y extends X {
          int b;
11
12
          YO {
13
              System.out.println("Inside Y's constructor");
14
15
      }
16
17
      class Z extends Y {
18
          int c;
19
          Z() {
20
              System.out.println("Inside Z's constructor");
21
22
      }
23
24
      public class MultilevelInheritance {
25
          public static void main(String[] args) {
26
              Z z = new Z();
27
              z.a = 10;
              z.b = 20;
28
              z.c = 30;
29
30
                                             Prepared By - Rifat Shahriyar
31
      }
```

Inside X's constructor Inside Y's constructor Inside Z's constructor

Method Overriding

```
class Base {
 3
            int a;
 4
            Base(int a) {
 5
                this.a = a;
 6
 7
   8
            void show() {
                System.out.println(a);
9
                                                    public class MethodOverride {
10
                                            28
       습}
11
                                                        public static void main(String[] args) {
                                            29
12
                                                             Child o = new Child( a: 10, b: 20);
                                            30
        class Child extends Base {
13
                                                             o.show();
                                            31
            int b;
14
                                                             Base b = o;
                                            32
15
                                                             b.show(); // will call show of Override
                                            33
            Child(int a, int b) {
16
                                            34
                super(a);
17
                                            35
                this.b = b;
18
19
20
            // the following method overrides Base class's show()
21
            Override // this is an annotation (optional but recommended)
22
            void show() {
23 0
                System.out.println(a + ", " + b);
24
25
26
```

Dynamic Method Dispatch

```
class P {
 3
    ©↓
    4
            void call() {
 5
                System.out.println("Inside P's call method");
 6
            }
 7
        }
        class Q extends P {
 8
 9 of ol
            void call() {
                System.out.println("Inside Q's call method");
10
11
            }
12
13
        class R extends 0 {
14 of
            void call() {
15
                System.out.println("Inside R's call method");
16
            }
17
        }
18
19
        public class DynamicDispatchTest {
20
            public static void main(String[] args) {
21
                P p = new P(); // object of type P
22
                Q q = new Q(); // object of type Q
23
                R r = new R(); // object of type R
24
                P x;
                               // reference of type P
25
                               // x refers to a P object
                x = p;
26
                             // invoke P's call
                x.call();
27
                x = q;
                               // x refers to a Q object
28
                            // invoke Q's call
                x.call();
                               // x refers to a R object
29
                x = r;
30
                x.call();
                               // invoke R's call
31
32
```

For practical example please refer to **FindAreas.java**

Abstract Class

- abstract class A
- contains abstract method *abstract method f()*
- No instance can be created of an abstract class
- The subclass must implement the abstract method
- Otherwise the subclass will be a abstract class too

Abstract Class



Anonymous Subclass



Using final with Inheritance class A { final void f() { System.out.println("This is a final method."); To prevent overriding } class B extends A { void f() { // Error! Can't override. System.out.println("Illegal!"); final class A { //... } To prevent inheritance // The following class is illegal. class B extends A { // Error! Can't subclass A //...

Local Variable Type Inference and Inheritance

- A superclass reference can refer to a derived class object in Java
- When using local variable type inference, the inferred type of a variable is based on the declared type of its initializer
 - Therefore, if the initializer is of the superclass type, that will be the inferred type of the variable
 - It does not matter if the actual object being referred to by the initializer is an instance of a derived class

Local Variable Type Inference and Inheritance

```
class A {
 1 0
                                                       19
                                                          2
            int a;
                                                       20
      <u>հ</u>}
 3
                                                       21
        class B extends A {
  22
            int b:
 5
                                                       23
      <u>6</u>}
                                                       24
 6
       class C extends B {
 7
                                                       25
            int c;
                                                       26
 8
       今}
 9
                                                       27
        public class InheritanceVarDemo {
10
                                                       28
            static A getObject(int type) {
11 @
                                                       29
                 switch(type) {
12
                     case 0: return new A();
13
                     case 1: return new B();
14
                     case 2: return new C();
15
                     default: return null;
16
17
18
```

For detail example please refer to InheritanceVarDemo.java

The inferred type is determined by the return type of getObject(), not by the actual type of the object obtained. Thus, all three variables will be of type A

Object Class

- 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
- This means that a reference variable of type Object can refer to an object of any other class
- Also, since arrays are implemented as classes, a variable of type Object can also refer to any array

Object's toString()

- The **toString()** method returns a string that contains a description of the object on which it is called
- Also, this method is automatically called when an object is output using println()
- Many classes override this method
- Doing so allows them to provide a description specifically for the types of objects that they create

Object's toString()

```
class Point {
 3
             int x, y;
 4
 5
             Point(int x, int y) {
 6
                 this.x = x;
 7
                 this.\mathbf{v} = \mathbf{v};
 8
 9
10
             a0verride
11
12 0
             public String toString() {
                 return "(" + x + ", " + y + ")";
13
14
15
        }
16
        public class ObjectTest {
17
             public static void main(String[] args) {
18
   Point p1 = new Point( x: 10, y: 20);
19
                 // without override toString() method the
20
                 // following will print something like this
21
                 // Point@3cd1a2f1
22
                 System.out.println(p1);
23
24
25
26
                        Prepared By - Rifat Shahriyar
```

Object's equals() and hashCode()

- == is a reference comparison, whether both variables refer to the same object
- Object's equals() method does the same thing
- String class override **equals()** to check contents
- If you want two different objects of a same class to be equal then you need to override equals() and hashCode() methods
 - hashCode() needs to return same value to work properly as keys in Hash data structures

Object's equals() and hashCode()

```
import java.util.HashMap;
 3
        import java.util.Objects;
 4
                                                               public class ObjectTest {
                                                       30
 5
                                                                   public static void main(String[] args) {
                                                      31
        class Point {
 6
                                                                       Point p1 = new Point( x: 10, y: 20);
                                                       32
            int x, y;
 7
                                                                       Point p2 = new Point( x: 10, y: 20);
                                                       33
            Point(int x, int y) {
 8
                                                                       System.out.println(p1.equals(p2));
                                                       34
                this.x = x;
 9
                                                                       System.out.println(p1 == p2);
                                                       35
                 this.y = y;
10
                                                                       HashMap m = new HashMap();
                                                       36
            }
11
                                                                       m.put(p1, "Hello");
                                                       37
12
                                                                       System.out.println(m.get(p2));
                                                       38
            ∂0verride
13
                                                       39
            public boolean equals(Object o) {
14 0
                                                       40
                 if (o == this) return true;
15
                                                       41
                 if (!(o instanceof Point)) {
16
                     return false;
17
18
                 Point p = (Point) o:
19
                 if (p.x == this.x && p.y == this.y) return true;
20
                 return false:
21
22
23
24
            ∂0verride
25 0
            public int hashCode() {
                return Objects.hash(x, y);
26
            }
27
28
```