Programming in Java

Dr. Nyein Aye Maung Maung
Dr. Eng (Ritsumeikan University, Japan)
Lecturer
Computer Engineering and Information Technology Dept.
Yangon Technological University

Course Schedule

Week	Topics
Week 1	Overview of JAVA, Data Types, Variables and Arrays
Week 2	Operators and Control Statements
Week 3	Classes and A closer look at Methods and Classes
Week 4	Inheritance, Overriding and Polymorphism
Week 5	Interfaces and Packages
Week 6	Exception Handling and Multi-threaded Programming
Week 7	String Handling
Week 8	Exploring java.lang and More utilities classes
Week 9	Java Collections Framework
Week 10	Java I/O
Week 11	Basic Graphical User Interface
Week 12	Event Handling
Week 13	Database Programming
Week 14	Applet and Networking

Lecture 3

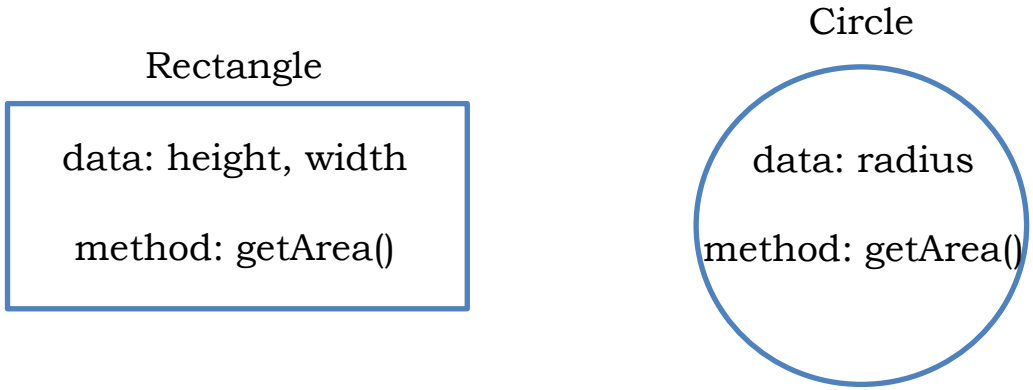
Classes, A Closer Look at Methods and Classes

Outline of Class

- Introduction to Classes
- A close look on Objects and Methods

Introduction to Class

- A class defines the properties and behaviors for objects
- Object-oriented programming (OOP) involves programming using objects.
  - An object represents an entity in the real world.
  - An object has a unique identity, state, and behavior.
- The state of an object (also known as its properties or attributes) is represented by data fields with their current values.
- The behavior of an object (also known as its actions) is defined by methods. To invoke a method on an object is to ask the object to perform an action.



Why Use Classes?

- Why not just primitives?

```
// student Alex
  String nameAlex;
  int idAlex;
// student david
  String nameDavid;
  int idDavid;
// student david
  String nameDavid2;
  int idDavid2;
```

500 students?  That Sucks!

Why Use Classes?

Student

Student1

Student2

Student3

… 497 Students

Defining Classes

```
class Box {

        double width;
        double height;
        double depth;
        /** construct a box object using default constructor*/
        Box(){
        }

        /* construct a box object using constructor parameterized*/
        Box(double w, double h, double d)
        {
                width=w;
                height=h;
                depth=d;
        }
        double volume( )
        {
                return width*height*depth;
        }
        }
```

Data fields

Constructors

Method

Sample Box Class

Constructors- Creating Objects

- Constructors are a special kind of method. They have three peculiarities:
  - A constructor must have the same name as the class itself
  - Constructors do not have a return type—not even void .
  - Constructors are invoked using the new operator when an object is created. Constructors play the role of initializing objects.
- Two types
  - Default Constructor
  - Parameterized Constructors
- A constructor is invoked to create an object using the **new** operator.

Constructors

```
public class CLASSNAME{
    CLASSNAME ( ) { } //Default
    CLASSNAME ([ARGUMENTS]) { } //Parameterized
}
```

```
Box(){}
Box(double w, double h, double d)
{
    width=w;
    height=h;
    depth=d;
}
```

```
//Constructing Objects Using Constructors
CLASSNAME obj1 = new CLASSNAME();
CLASSNAME obj2 = new CLASSNAME([ARGUMENTS])
```

```
Box b1=new Box();
Box b2=new Box(2,3,4)
```

Accessing an Object's Data and Methods

- An object's member refers to its data fields and methods.
- After an object is created, its data can be accessed and its methods invoked using the dot operator (.), als o known as the object member access operator.
  - **objectRefVar.dataField** references a data field in the object.
  - **objectRefVar.method(arguments)** invokes a method on the object

Example 3.1. Create two box objects mybox1 and mybox2 using default constructor and parameterized constructor respectively. Assign dimensions of mybox1 and compute volumes of each box.

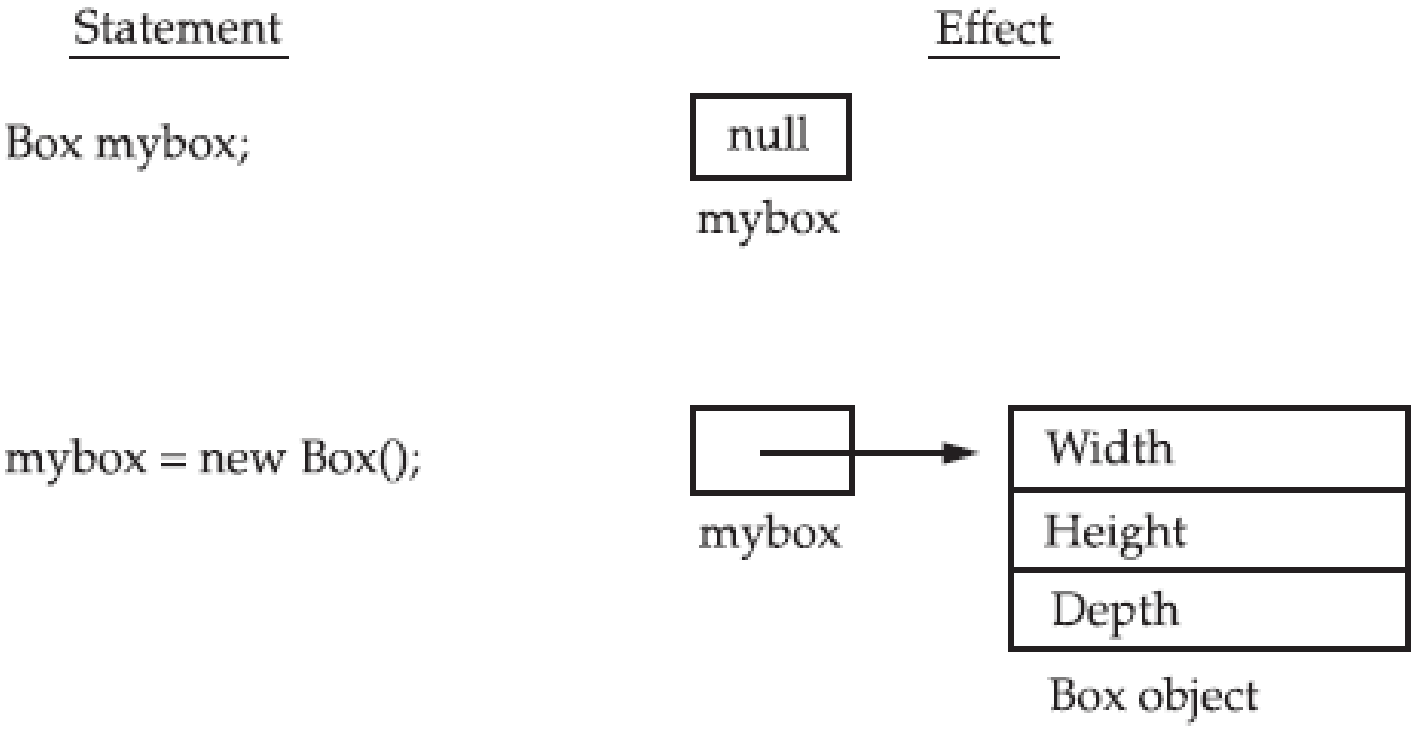
```
class BoxDemo {
    public static void main(String args[]) {
        Box mybox1 = new Box(); //Create box object using default constructor
        Box mybox2 = new Box(3,6,9); //Using parameterized constructor
        double vol;
        // assign values to mybox1's instance variables
        mybox1.width = 10;
        mybox1.height = 20;
        mybox1.depth = 15;
        // compute volume of first box
        vol = mybox1.width * mybox1.height * mybox1.depth;
        System.out.println("Volume of box 1 is " + vol);
        // compute volume of second box using object's method
        System.out.println("Volume of box 2 is " + mybox2.volume());
    }
}
```

Accessing fields:
object.**FieldName**

Accessing method:
object.**MethodName**

Output
Volume of box 1 is 3000.0
Volume of box 2 is 162.0

A closer look at new



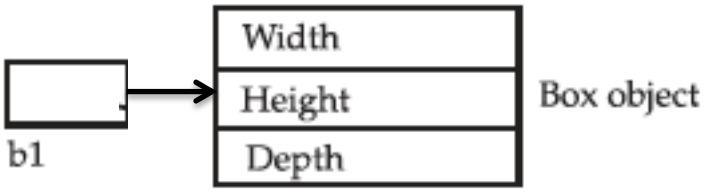
- class → logical construct
- object → physical reality
- The object's location is called a reference

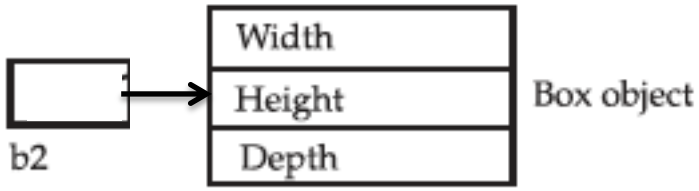
Assigning Object Reference Variables

- == operator compares the references

  Box b1=new Box (2,3,4);

  Box b2= new Box(2,3,4);



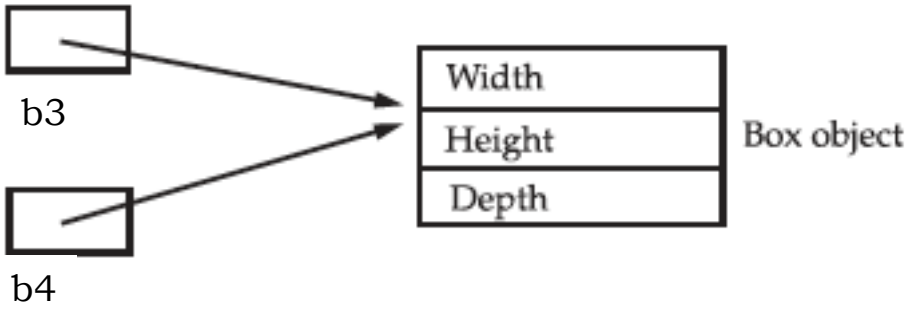


Does b1==b2? → No

Box b3=new Box();

Box b4=b3;

Does b3==b4? → Yes



- b3 and b4 refer to the same object
- Any changes made to the object through b3 will affect the object to which b3 is referring

Example 3.2. Demonstrate Object Reference Variables

```
. class BoxDemo {
public static void main(String args[]) {
        Box box1=new Box (2,3,4);
        Box box2= new Box(2,3,4);
        Box box3=box1;
        if(box1==box2)
        System.out.println("Memory references of Box 1 and box 2 are same." );
        if(box1==box3)
        System.out.println("Memory references of Box 1 and box 3 are same." );
        System.out.println("Box 1 : "+box1.width+"\t"+box1.height+"\t"+box1.depth);
        System.out.println("Box 2 : "+box2.width+"\t"+box2.height+"\t"+box2.depth);
        System.out.println("Box 3 : "+box3.width+"\t"+box3.height+"\t"+box3.depth);
        // Update box1's instance variables
        box1.width = 10;
        box1.height = 20;
        box1.depth = 15;
        System.out.println("After updating box 1:\n");
        System.out.println("Box 1 : "+box1.width+"\t"+box1.height+"\t"+box1.depth);
        System.out.println("Box 2 : "+box2.width+"\t"+box2.height+"\t"+box2.depth);
        System.out.println("Box 3 : "+box3.width+"\t"+box3.height+"\t"+box3.depth);
}
}
```

Output

Memory references of Box 1 and box 3 are same.
Box 1 : 2.0 3.0 4.0
Box 2 : 2.0 3.0 4.0
Box 3 : 2.0 3.0 4.0
After updating box 1:

Box 1 : 10.0 20.0 15.0
Box 2 : 2.0 3.0 4.0
Box 3 : 10.0 20.0 15.0

Introducing Methods

- A method is a program module that contains a series of statements that carry out a task.
- Any class can contain an unlimited number of methods, and each method can be called an unlimited number of times

```
type  name (parameter-list) {
// body of method
}
```

- type: specifies the type of data returned by the method
  No return → void

```
class Box {
        double width;
        double height;
        double depth;
        // Method that computes and returns volume
        //No parameter, return type 'double'
        double volume() {
                return width * height * depth;
        }
        // Method that sets dimensions of box, no return
        //Three parameters, no return
        void setDim(double w, double h, double d) {
                width = w; height = h;
                depth = d;
        }
        }
```

Sample Box Class with Methods

Example 3.3. Create two box objects mybox1 and mybox2 using default con structor and set dimensions of them using setDim method of Box class. Fina lly, compute volumes of each box.

```
class BoxDemo5 {
public static void main(String args[]) {
      Box mybox1 = new Box();
      Box mybox2 = new Box();
      double vol;
      // initialize each box
      mybox1.setDim(10, 20, 15);
      mybox2.setDim(3, 6, 9);
      // get volume of first box
      vol = mybox1.volume();
      System.out.println("Volume is " + vol);
      // get volume of second box
      vol = mybox2.volume();
      System.out.println("Volume is " + vol);
}
}
```

Static Variables, Constants, and Methods

- **Static Variable**
  - *Shared by all objects of the class*
  - It is a variable which belongs to the class and not to object(instance)
  - Static variables are initialized only once, at the start of the execution. These variables will be initialized first, before the initialization of any instance variables
  - A single copy to be shared by all instances of the class
  - A static variable can be accessed directly by the class name and doesn't need any object
- **Static Methods**
  - It is a method which belongs to the class and not to the object(instance)
  - A static method can access only static data. It can not access non-static data (instance variables)
  - A static method can call only other static methods and can not call a non-static method from it.
  - A static method can be accessed directly by the class name and doesn't need any object
  - A static method cannot refer to "this" or "super" keywords in anyway

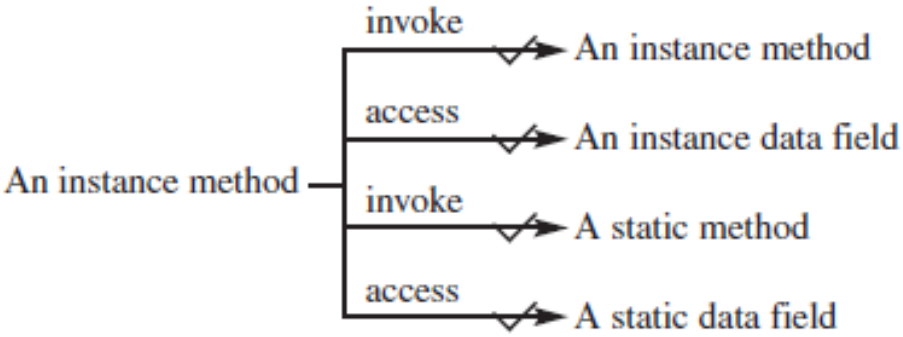
Static Variables, Constants, and Methods

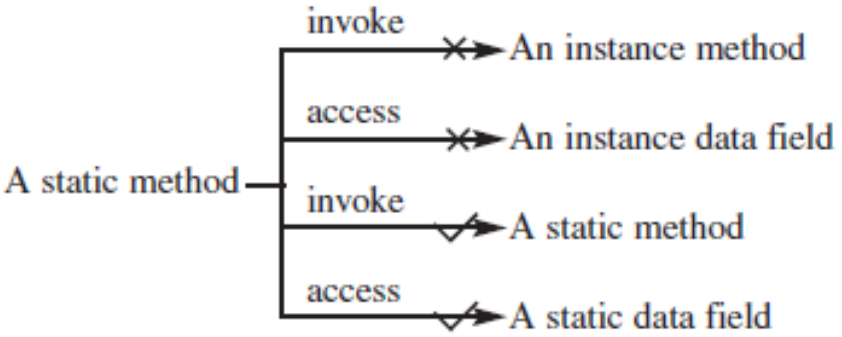
```
static int numberOfObjects;

static int getNumberObjects() {
  return numberOfObjects;
}
```

Declare static variable

Declare static method

- Static or Instance, how to decide?
  - A variable or method that is dependent on a specific instance of the class should be an instance variable or method.
  - A variable or method that is not dependent on a specific instance of the class should be a static variable or method






```
1  public class A {
2    int i = 5;
3    static int k = 2;
4
5    public static void main(String[] args) {
6      int j = i; // Wrong because i is an instance variable
7      m1(); // Wrong because m1() is an instance method
8    }
9
10   public void m1() {
11     // Correct since instance and static variables and methods
12     // can be used in an instance method
13     i = i + k + m2(i, k);
14   }
15
16   public static int m2(int i, int j) {
17     return (int)(Math.pow(i, j));
18   }
19 }
```

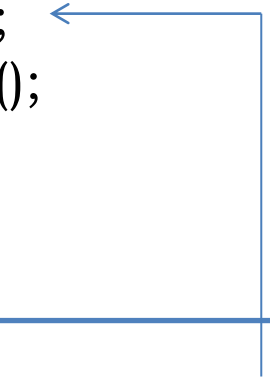
```
1  public class A {
2    int i = 5;
3    static int k = 2;
4
5    public static void main(String[] args) {
6      A a = new A();
7      int j = a.i; // OK, a.i accesses the object's instance variable
8      a.m1(); // OK. a.m1() invokes the object's instance method
9    }
10
11   public void m1() {
12     i = i + k + m2(i, k);
13   }
14
15   public static int m2(int i, int j) {
16     return (int)(Math.pow(i, j));
17   }
18 }
```

Example 3.4. Demonstrate the use of Static and Instance Variables and Methods

```
class Student {
      int a; //initialized to zero
      static int b; //initialized to zero only when class is loaded, not for
      each object created.

        Student(){
         //Constructor incrementing static variable b
         b++;
        }

         public void showData(){
            System.out.println("Value of a = "+a);
            System.out.println("Value of b = "+b); //can access static variable
                                                     from non-static method
         }
    public static void increment(){
          a++;
    }

}
```

Error block: static method cannot access instance variable

```
public class Demo{
   public static void main(String args[]){
     Student s1 = new Student();
     s1.showData();
     Student s2 = new Student();
     s2.showData();
     Student.b++;
     s1.showData();


}
}
```



directly access by the class name

Example 3.5: Demonstrate the use of static variable using Circle Class

```
1  public class Circle2 {
2    /** The radius of the circle */
3    double radius;
4
5    /** The number of the objects created */
6    static int numberOfObjects = 0;
7
8    /** Construct a circle with radius 1 */
9    Circle2() {
10     radius = 1.0;
11     numberOfObjects++;
12   }
13
14   /** Construct a circle with a specified radius */
15   Circle2(double newRadius) {
16     radius = newRadius;
17     numberOfObjects++;
18   }
19
20   /** Return numberOfObjects */
21   static int getNumberOfObjects() {
22     return numberOfObjects;
23   }
24
25   /** Return the area of this circle */
26   double getArea() {
27     return radius * radius * Math.PI;
28   }
29 }
```

```java
public class  TestCircle2 {
  /** Main method */
  public static void main(String[] args) {
    // Create c1
    Circle2 c1 = new Circle2();

    // Display c1 BEFORE c2 is created
    System.out.println("Before creating c2");
    System.out.println("c1 is : radius (" + c1.radius +
      ") and number of Circle objects (" +
      c1.numberOfObjects + ")");

    // Create c2
    Circle2 c2 = new Circle2(5);

    // Change the radius in c1
    c1.radius = 9;

    // Display c1 and c2 AFTER c2 was created
    System.out.println("\nAfter creating c2 and modifying " +
      "c1's radius to 9");
    System.out.println("c1 is : radius (" + c1.radius +
      ") and number of Circle objects (" +
      c1.numberOfObjects + ")");
    System.out.println("c2 is : radius (" + c2.radius +
      ") and number of Circle objects (" +
      c2.numberOfObjects +")");
  }
}
```

```
c1 is : radius (1.0) and number of Circle objects (1)

After creating c2 and modifying c1's radius to 9
c1 is : radius (9.0) and number of Circle objects (2)
c2 is : radius (5.0) and number of Circle objects (2)
```

Quiz I

1. What is wrong with each of the following programs

```
1  public class ShowErrors {
2    public static void main(String[] args) {
3      ShowErrors t = new ShowErrors(5);
4    }
5  }
```

```
1  public class ShowErrors {
2    public static void main(String[] args) {
3      ShowErrors t = new ShowErrors();
4      t.x();
5    }
6  }
```

```
1  public class ShowErrors {
2    public void method1() {
3      Circle c;
4      System.out.println("What is radius "
5        + c.getRadius());
6      c = new Circle();
7    }
8  }
```

```
 1  public class ShowErrors {
 2    public static void main(String[] args) {
 3      C c = new C(5.0);
 4      System.out.println(c.value);
 5    }
 6  }
 7
 8  class C {
 9    int value = 2;
10  }
```

2. What is wrong with the following code

```
 1  class Test {
 2    public static void main(String[] args) {
 3      A a = new A();
 4      a.print();
 5    }
 6  }
 7
 8  class A {
 9    String s;
10
11    A(String newS) {
12      s = newS;
13    }
14
15    public void print() {
16      System.out.print(s);
17    }
18  }
```

3. Can you invoke an instance method or reference an instance variable from a static method? Can you invoke a static method or reference a static variable from an instance method? What is wrong in the following code?

```
1  public class C {
2    public static void main(String[] args) {
3      method1();
4    }
5
6    public void method1() {
7      method2();
8    }
9
10   public static void method2() {
11     System.out.println("What is radius " + c.getRadius());
12   }
13
14   Circle c = new Circle();
15 }
```

- **Exercise 3.1.** Write a JAVA class named “Box” which has three double variables for the dimension of the box: width, height and depth, default constructor, three argument constructors which will initialize the variables values, the method “volume” which will calculate the volume of boxes and return the double value, and the method “setDimension” which will be used by the objects to set dimensions of the box. Write another JAVA class “BoxDemo” to create two objects, one by using default constructor, another by using argument constructor. And then set the dimension of the first object by calling setDimension method and compute volumes of boxes.

```
class Box{

double width;
double height;
double depth;
/** construct a box object */
Box(){
}
/* construct a box object */
Box(double w, double h, double d)
{
     width=w;
     height=h;
     depth=d;
}
```

```
double volume()
{
    return this.width*this.height*this.depth;
}
void setDimension(double w, double h, double d)
{
    width=w;
    height=h;
    depth=d;
}
}

class BoxDemo2
{
    public static void main(String args[])
    {
        Box mybox1 = new Box();
        Box mybox2 = new Box(3, 6, 9);
        double vol;
        mybox1.setDimension(2, 3, 4);
        vol = mybox1.volume();
        System.out.println("Volume of box 1 is " + vol);
        vol = mybox2.volume();
        System.out.println("Volume of box 2 is " + vol);
    }
}
```

The ‘this’ Keyword

- Can be used inside any method to refer to the current object

```
// A redundant use of this.
Box(double w, double h, double d) {
  this.width = w;
  this.height = h;
  this.depth = d;
}
```

Perfectly correct, but redundant

```
// Use this to resolve name-space collisions.
Box(double width, double height, double depth) {
  this.width = width;
  this.height = height;
  this.depth = depth;
}
```

To resolve name collisions

Exercise 3.2. Write a JAVA class "Stack" which defines an integer stack that can hold 10 values, push and pop methods. Write another JAVA class which will create the two objects of Stack Class , calling push method to assign the integer values 1 to 10 and 11 to 20 to the Stack and calling pop method to print out the data from stacks.

```
class Stack {
        int stck[] = new int[10];
        int tos;
        Stack() {
                tos = -1;
        }
        void push(int item) {
                if(tos==9)
                System.out.println("Stack is full.");
                else
                stck[++tos] = item;
        }
        int pop(){
                if(tos < 0) {
                System.out.println("Stack underflow.");
                return 0;
                }
                else
                return stck[tos--];
        }
}
```

Initialize top-of-stack

Push an item onto the stack

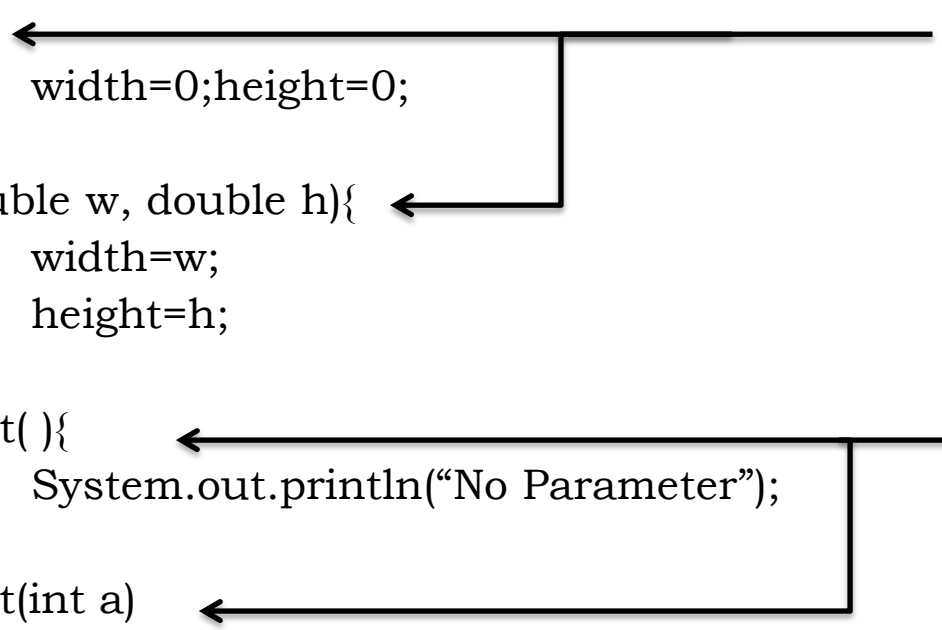
Pop an item from the stack

```
class TestStack {
    public static void main(String args[]) {
        Stack mystack1 = new Stack();
        Stack mystack2 = new Stack();
        // push some numbers onto the stack
        for(int i=0; i<10; i++) mystack1.push(i);
        for(int i=10; i<20; i++) mystack2.push(i);
        // pop those numbers off the stack
        System.out.println("Stack in mystack1:");
        for(int i=0; i<10; i++)
        System.out.println(mystack1.pop());
        System.out.println("Stack in mystack2:");
        for(int i=0; i<10; i++)
        System.out.println(mystack2.pop());
    }
}
```

Method Overloading

- Define two or more methods within the same class that share the same name, as long as their parameter declarations are different.

```
class Box{
        double width, height;
        Box(){
                width=0;height=0;
        }
        Box(double w, double h){
                width=w;
                height=h;
        }
        void test( ){
                System.out.println("No Parameter");
        }
        void test(int a)
        {
                System.out.println("a="+a);
        }
}
```



Argument Passing

- Call-by-value
  - copies the value of an argument into the formal parameter of the subroutine
- Call-by-reference
  - a reference to an argument (not the value of the argument) is passed to the parameter
  - changes made to the parameter will affect the argument used to call the subroutine

Example 3.6: Demonstrate call-by-value parameter passing

```
// Primitive types are passed 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);
  }
}
```

Example 3.7: Demonstrate call-by-reference parameter passing

```
// Objects are passed by reference.
class Test {
    int a, b;
    Test(int i, int j) {
        a = i;
        b = j;
    }
    // pass an object
    void meth(Test o) {
        o.a *= 2;
        o.b /= 2;
    }
}
```

```
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("After call:"+
    ob.a + " " + ob.b);
    }
}
```

Returning Objects

- A method can return any type of data, including class types that you create

```
// Returning an object.
class Test {
      int a;
      Test(int i) {
            a = i;
      }
      Test incrByTen() {
            Test temp = new Test(a+10);
            return temp;
      }
}
```

```
class RetOb {
      public static void main(String args[]) {
      Test ob1 = new Test(2);
      Test ob2;
      ob2 = ob1.incrByTen();
      System.out.println("ob1.a: " + ob1.a);
      System.out.println("ob2.a: " + ob2.a);
      ob2 = ob2.incrByTen();
      System.out.println("ob2.a after
      second increase: "
      + ob2.a);
      }
}
```

Recursion

- Process of defining something in terms of itself
- A method that calls itself is said to be recursive.

```
returntype methodname(){
//code to be executed
        methodname();//calling same method
}
```

Exercise 3.3. Write a program that computes factorial of a number.

```
class Factorial {
// this is a recursive method
int fact(int n) {
      int result;
      if(n==1) return 1;
      result = fact(n-1) * n;
      return result;
      }
}
class Recursion {
      public static void main(String args[]) {
      Factorial f = new Factorial();
      System.out.println("Factorial of 3 is " + f.fact(3));
      System.out.println("Factorial of 4 is " + f.fact(4));
      System.out.println("Factorial of 5 is " + f.fact(5));
}
}
```

Nested and Inner Classes

- Nested Class
  - A class within another class
  - Two types
    - Non-static nested class / Inner class
      - has access to all of the variables and methods of its outer class
    - Static nested class
      - A static nested class is one that has the **static** modifier applied
      - it must access the members of its enclosing class through an object.

Example 3.8: Demonstrate Inner Class

```
class Outer_Demo {
   int num;
   class Inner_Demo {                    //Inner Class
      public void print() {
          System.out.println("This is an inner class");
      }
   }
   void display_Inner() {
      Inner_Demo inner = new Inner_Demo();
      inner.print();
   }
} public class My_class {

   public static void main(String args[]) {
      // Instantiating the outer class
      Outer_Demo outer = new Outer_Demo();
      // Accessing the display_Inner() method.
      outer.display_Inner();
   }
}
```

Example 3.9: Demonstrate Inner Class and scope of its variables

```
class Outer {
      int outer_x = 100;
      void test() {
      Inner inner = new Inner();
            inner.display();
      }
      // this is an inner class
      class Inner {
            int y = 10; // y is local to Inner
            void display() {
            System.out.println("display: outer_x = " + outer_x);
            }
      }
      void showy() {
            System.out.println(y); // error, y not known here! Outside scope of y
            }
      }
      class InnerClassDemo {
      public static void main(String args[]) {
            Outer outer = new Outer();
            outer.test();
      }
}
```

Quiz II

1. Show the printout of the following codes.

```
public class Test {
  public static void main(String[] args) {
    int[] a = {1, 2};
    swap(a[0], a[1]);
    System.out.println("a[0] = " + a[0]
      + " a[1] = " + a[1]);
  }

  public static void swap(int n1, int n2) {
    int temp = n1;
    n1 = n2;
    n2 = temp;
  }
}
```

(a)

```
public class Test {
  public static void main(String[] args) {
    int[] a = {1, 2};
    swap(a);
    System.out.println("a[0] = " + a[0]
      + " a[1] = " + a[1]);
  }

  public static void swap(int[] a) {
    int temp = a[0];
    a[0] = a[1];
    a[1] = temp;
  }
}
```

(b)

2. Show the printout of the following codes.

```
public class Test {
  public static void main(String[] args) {
    T t = new T();
    swap(t);
    System.out.println("e1 = " + t.e1
      + " e2 = " + t.e2);
  }

  public static void swap(T t) {
    int temp = t.e1;
    t.e1 = t.e2;
    t.e2 = temp;
  }
}

class T {
  int e1 = 1;
  int e2 = 2;
}
```

(c)

```
public class Test {
  public static void main(String[] args) {
    T t1 = new T();
    T t2 = new T();
    System.out.println("t1's i = " +
      t1.i + " and j = " + t1.j);
    System.out.println("t2's i = " +
      t2.i + " and j = " + t2.j);
  }
}

class T {
  static int i = 0;
  int j = 0;

  T() {
    i++;
    j = 1;
  }
}
```

(d)

Summary

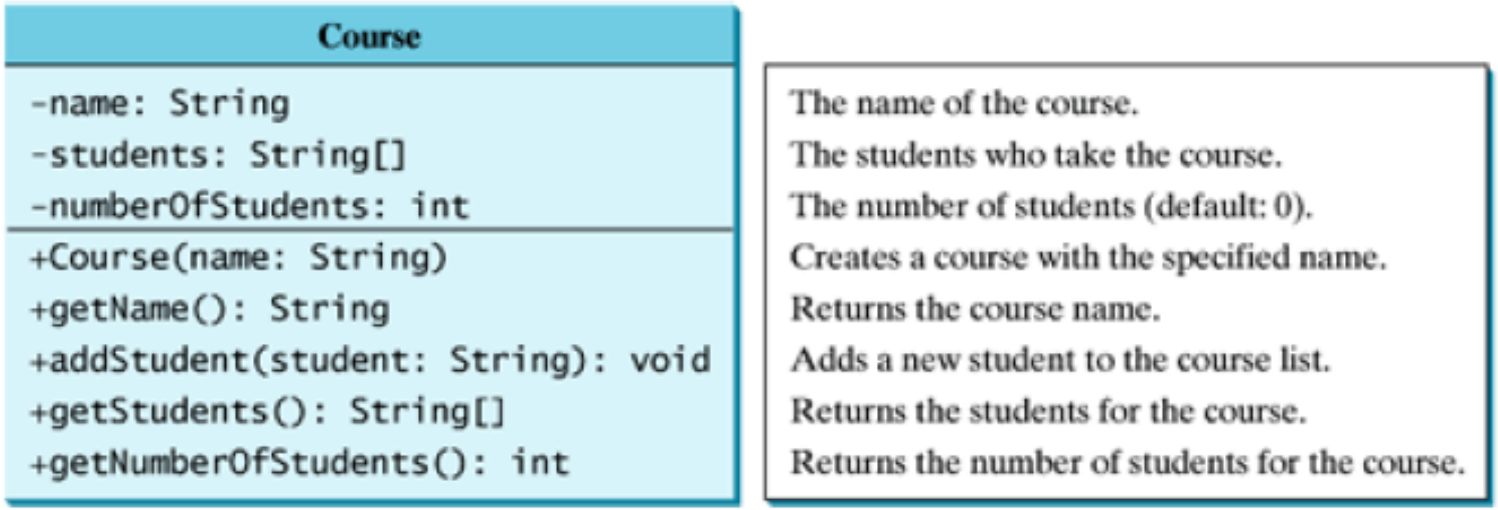
- A class is a template for objects. It defines the generic properties of objects, and provides constructors for creating objects and methods for manipulating them.
- An object is an instance of a class. You use the new operator to create an object, and the dot (.) operator to access members of that object through its reference variable.
- An instance variable or method belongs to an instance of a class. Its use is associated with individual instances. A static variable is a variable shared by all instances of the same class. A static method is a method that can be invoked without using instances.
- Every instance of a class can access the class's static variables and methods. However, it is better to invoke static variables and methods using ClassName.variable and ClassName.method for clarity.
- The keyword this can be used to refer to the calling object.

Practical Assignments

1. Creat a class named ‘Course’ which contains the following data fields and methods.

Then, write a class to test the Course class which performs the followings.

- Create two courses using the constructor Course(String name) by passing a course name.
- Add students to the course using the addStudent (String student) method
- And return all the students for the course using the getStudents() method



2. Design a class named Fan to represent a fan. The class contains:

- Three constants named SLOW, MEDIUM, and FAST with values 1, 2, and 3 to denote the fan speed.
- An int data field named speed that specifies the speed of the fan (default SLOW).
- A boolean data field named on that specifies whether the fan is on (default false).
- A double data field named radius that specifies the radius of the fan (default 5).
- A string data field named color that specifies the color of the fan (default blue).
- A no-arg constructor that creates a default fan.
- A four argument constructor that creates a fan obj with user defined speed, radius, color and fan status.
- A method named toString() that returns a string description for the fan. If the fan is on, the method returns the fan speed, color, and radius in one combined string. If the fan is not on, the method returns fan color and radius along with the string "fan is off" in one combined string.

Implement the class. Write a test program that creates two Fan objects. Assign maximum speed, radius 10, color yellow, and turn it on to the first object. Assign medium speed, radius 5, color blue, and turn it off to the second object. Display the objects by invoking their toString method.

Next Week Lecture

Inheritance, Method Overriding and Polymorphism

Thank you!