Object and Classes

B.Tech. (IT), Sem-5,
Core Java Technology (CJT)

Dharmsinh Desai University
Prof. (Dr.) H B Prajapati

1

Lecture-1

2

Object based vs Object Oriented

- Object oriented programming languages follow all concepts belonging to OOP
  - C++, Java
- Object-based language doesn't support all the features of OOPs like Polymorphism and Inheritance
  - Object-based language has in-built object
  - Javascript, VB etc

3

OOP Principles (Encapsulation, Inheritance, & Polymorphism)

- Abstraction
  - Interface is exposed rather than implementation
  - It is there in Procedural approach (e.g., we use printf() without knowing implementation details)
- Encapsulation
  - Data and functions (services) are bound together and data are made private (the state is encapsulated or hidden)
  - Example: Various parts in a car are encapsulated in their own units and have only interfaces to connect one another. E.g., accelerating does not affect light system.
  - Private things can change without affecting to service user
- Inheritance
  - Derive characteristics from another class
- Polymorphism
  - A single method form (signature) can be used on different classes.

4

Object and classes

- Object
  - Instance of a class
  - Separate copy of data members
- Class
  - Blue print
  - Specification of objects
  - Single copy of code

5

Is class required at runtime?

- Example:
  - Box b=new Box();
- We require both object and class at runtime as method (even constructor) specification is in class (shared by all objects)
- Each instance/object gets its own copy of variable.
  - E.g., Alto 800 cars, each has different state (e.g., amount of fuel in tank, distance travelled, etc.)

6

Using javac and java

- Test.java

```
class A{
    public static void main(String[] args){
        System.out.println("Hello");
    }
}
```

- Compile: $javac Test.java
- Run: $java Test
- To pass command line arguments: $java Test 1 2 3
- To access 1, 2, and 3, we can use Integer.parseInt()

7

File naming convention

- A single file can contain multiple Java classes, but can contain only one public class
  - Name of public class should be the name of the file
- If a file does not contain any public class, then any name can be used for a file.

8

Multiple classes in a single file

- Test.java

```
class A{}
class B{}
class C{}
```

- Compile using $javac Test.java
- It produces three .class files
  1. A.class
  2. B. class
  3. C. class

9

Lecture-2

10

Creating an object

- Creating an anonymous object:

```
new Circle();
```

- Creating an object and holding it in a reference:

```
Circle c=new Circle();
```

OR

```
Circle c;
C=new Circle();
```

11

Creating a String object

- String msg="Hello";
- String literals are String objects, implicitly created by Java

12

Data members

- Data members are declared in the class
- They can be initialized in the class
- They can also be initialized via constructor

13

Using constructor

- Constructor initializes an object of a class
- Java requires a constructor for every class
- If we do not provide any constructor, default constructor (no argument) is provided by Java
- Syntax:
  - Box b=new Box(10,20,30);
- Call a method without object reference
  - new Box().getVolume();

14

static scope

```
class Box{
    static{
        System.out.println("Static Block: Box");
    }
}
class BoxMain{
    public static void main(String[] args){
        System.out.println("Main: BoxMain");
        Box b;
    }
}
```

15

static scope

```
class Box{
    static{
        System.out.println("Static Block: Box");
    }
}
class BoxMain{
    public static void main(String[] args){
        System.out.println("Main: BoxMain");
        Box b;
        b=new Box();
    }
}
```

16

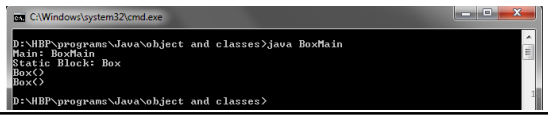
static scope (creating multiple objects)

```
class Box{
    static{
        System.out.println("Static Block: Box");
    }
    Box(){
        System.out.println("Box()");
    }
}
```

17

static scope (creating multiple objects)

```
class BoxMain{
    public static void main(String[] args){
        System.out.println("Main: BoxMain");
        Box b1,b2;
        b1=new Box();
        b2=new Box();
    }
}
```



Assigning objects versus assigning primitive types

```
int a=10;
int b=20;
a=b;
b=30;
```

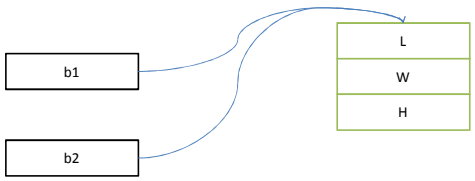
What will be the values of a and b?

```
Box b1=new
    Box(10,20,30);
Box b2;
b2=b1;
b2.setLength(20);
```

What will be the values of L, W, H for b1 and b2

19

Object reference



20

Method

- Local variables vs data members (instance variables or fields)

```
class Test{
    int i;
    void test(){
        int i;
        i=10;
    }
}
```

- In i=10; i refers to the i defined in the method, not defined in the class.

21

this pointer

- We can use this pointer to resolve ambiguity between parameter and member data, e.g., in constructor

```
Box(int l, int w, int h){
    this.l=l;
    this.w=w;
    this.h=h;
}
```

- To pass object itself to a method of some other class.
- E.g.,
  - Box has decorate() method
  - Painter class has paint(Box) method
  - From decorate() method, box object can be passed to Painter object using this pointer.

22

Using getter/setter methods

- Accessor method:
  - getter method
- Mutator method
  - setter method

23

Using getter/setter methods

- Example:

```
class Box{
    int width;
    public void setWidth(width){
        this.width=width;
    }
    public int getWidth(){
        return width;
    }
}
```

24

Passing objects/values to a method

```
class Box{
    int length,width,height;
    Box(){
        length=width=height=1;
    }
    void setLength(int length){
        this.length=length;
    }
    void setWidth(int width){
        this.width=width;
    }
```

25

Passing objects/values to a method

```
    void setHeight(int height){
        this.height=height;
    }
    public String toString(){
        return "Box: "+length+"X"+width+"X"+height;
    }
}
```

26

Passing objects/values to a method

```
class Test{
    public static void main(String[] args){
        Box b1=new Box();
        int i=1;
        test(i);
        System.out.println("i="+i);
        testBox(b1);
        System.out.println(b1);
    }
    public static void test(int i){
        i=10;
    }
```

27

Passing objects/values to a method

```
    public static void testBox(Box b){
        b.setLength(10);
    }
}
```

28

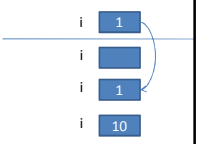
Lecture-3

29

Pass-by-value

- Primitive Data Types are passed by values
- When the method is called, the value of each argument is copied (assigned) to its corresponding formal parameter
- The method only gets a copy of the variable's value

```
int i=1;
test(i)
public static void test(int i){
    i=10;
}
```

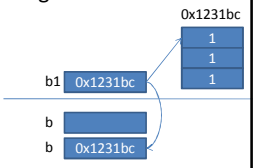


30

Pass-by-reference

- Objects are passed by reference
- When an object is passed to a method, the reference value is copied to the corresponding formal parameter, not actual object.

```
testBox(b1);
public static void testBox(Box b){
    b.setLength(10);
}
```



31

How to swap two objects?

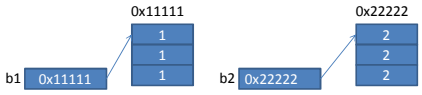
- Let's write swap() method and use it

```
Box b1=new Box(1,1,1);
Box b2=new Box(2,2,2);
swap(b1,b2);
static void swap(Box b1, Box b2){
    Box temp;
    temp=b1;
    b1=b2;
    b2=temp;
}
```

32

How to swap two objects?

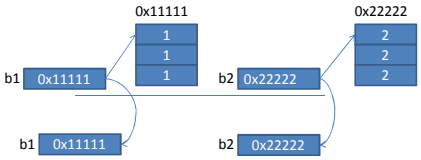
```
Box b1=new Box(1,1,1);
Box b2=new Box(2,2,2);
```



33

How to swap two objects?

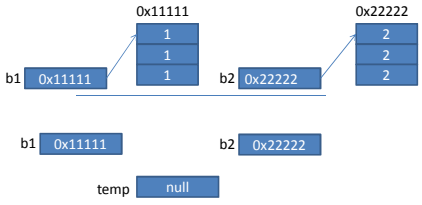
```
swap(b1,b2);
static void swap(Box b1, Box b2){
```



34

How to swap two objects?

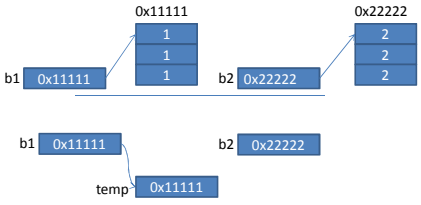
```
static void swap(Box b1, Box b2){
    Box temp;
```



35

How to swap two objects?

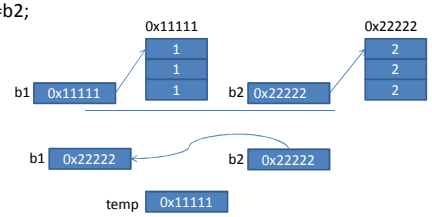
```
static void swap(Box b1, Box b2){
    Box temp;
    temp=b1;
```



36

How to swap two objects?

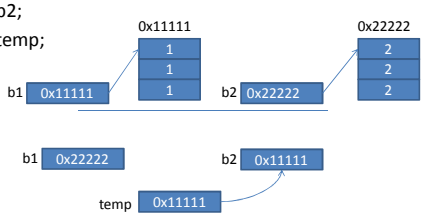
```
static void swap(Box b1, Box b2){
    Box temp;
    temp=b1;
    b1=b2;
```



37

How to swap two objects?

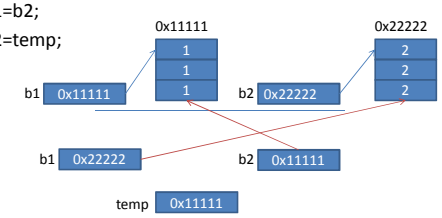
```
static void swap(Box b1, Box b2){
    Box temp;
    temp=b1;
    b1=b2;
    b2=temp;
```



38

How to swap two objects?

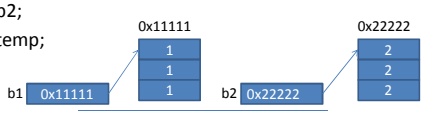
```
static void swap(Box b1, Box b2){
    Box temp;
    temp=b1;
    b1=b2;
    b2=temp;
```



39

It is incorrect solution !!!

```
static void swap(Box b1, Box b2){
    Box temp;
    temp=b1;
    b1=b2;
    b2=temp;
}
```



40

Correct solution

```
static void swap(Box b1, Box b2){
    Box temp=new Box();
    temp.l=b1.l; temp.w=b1.w; temp.h=b1.h;
    b1.l=b2.l;     b1.w=b2.w;     b1.h=b2.h;
    b2.l=temp.l; b2.w=temp.w; b2.h=temp.h;
}
```

41

Access modifiers

- Static
- Public
- Private
- Protected
- none (package)

42

Access modifiers

- static
  - Class-wide information
  - Can be used for data, methods, and even class
- public
  - All can access
  - Can be used for data, methods, and even class
- private
  - Only class members can access
  - Can be used for data, methods

43

Access modifiers

- Protected
  - Classes of the same package and subclass in any package can access
  - Can be used for data, methods
- none (package)
  - Only classes in same package can access
  - Can be used for data, methods, and even class

44

Modifier: private

- Example

```
class Box{
    private int length,width,height;
}
class Test{
    public static void main(String[] args){
        Box b1=new Box();
        b1.length=10;
    }
}
```

45

Modifier: private

```
class Box{
    private int length,width,height;
    Box(){
        length=width=height=1;
    }
    Box(int l, int w, int h){
        length=l;
        width=w;
        height=h;
    }
```

46

Modifier: private

```
    Box(Box b){
        this.length=b.length;
        this.width=b.width;
        this.height=b.height;
    }
}
class Test{
    public static void main(String[] args){
        Box b1=new Box();
        Box b2=new Box(10,10,10);
        Box b3=new Box(b2);
    }
}
```

This object can access private data of another object of the same class

47

Method overloading

- Method overloading allows us to use the same name for methods, performing similar task.
- Input arguments should be different.

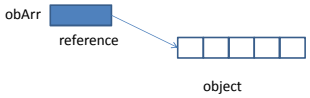
```
class Painter{
    public static void paint(Grill g){
        //paint the grill g using oil paint color with brush
    }
    public static void paint(Wall w){
        //paint the wall w using oil plastic paint color with roller
    }
}
```

48

How to create an array

- Array is an object

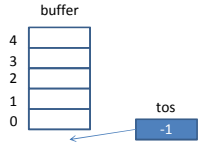
int[] obArr;
obArr=new int[5];



49

Implement a Stack with overflow and underflow condition

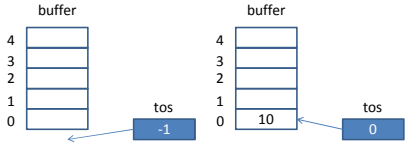
- A stack at initial condition…



50

Implement a Stack with overflow and underflow condition

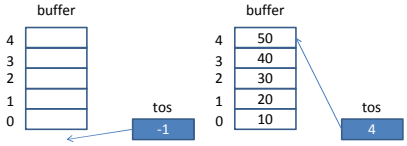
- push(10) operation



51

Implement a Stack with overflow and underflow condition

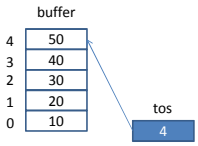
- push() operations, four more times…



52

Implement a Stack with overflow and underflow condition

- Now, no more push() are possible…



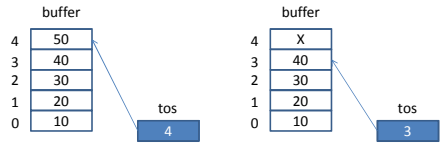
- If we call push(), we should generate stack overflow error…
- Condition?

If(tos==(buffer.length-1))

53

Implement a Stack with overflow and underflow condition

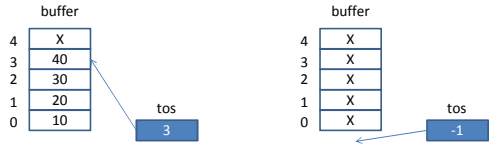
- pop() operation



54

Implement a Stack with overflow and underflow condition

- pop() operations, four more times



- Now, no more pop() operation is possible…
- But, if we perform pop(), we should get stack underflow error.
- Condition

If(tos<0)

55

Stack

```
class Stack{
    int[] buffer;
    int tos;
    public Stack(int size){
        buffer=new int[size];
        tos=-1;
    }
```

56

Stack

```
    public void push(int element){
        if(tos==(buffer.length-1)){
            System.out.println("Error: Stack Overflow");
        }
        else
            buffer[++tos]=element;
    }
```

57

Stack

```
    public int pop(){
        if(tos<0){
            System.out.println("Error: Stack Underflow");
            return -1;
        }
        else
            return buffer[tos--];
    }
}
```

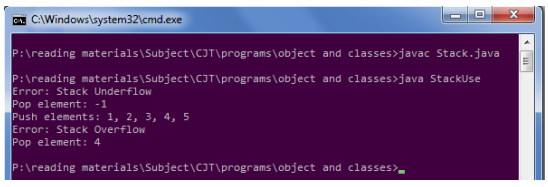
58

Use the Stack

```
class StackUse{
    public static void main(String[] args){
        Stack stack=new Stack(4);
        System.out.println("Pop element: "+stack.pop());
        System.out.println("Push elements: 1, 2, 3, 4, 5");
        stack.push(1);
        stack.push(2);
        stack.push(3);
        stack.push(4);
        stack.push(5);
        System.out.println("Pop element: "+stack.pop());
    }
}
```

59

Use the Stack



60

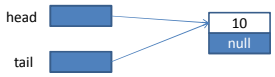
Lecture-4

61

Implement a Queue

- An empty queue

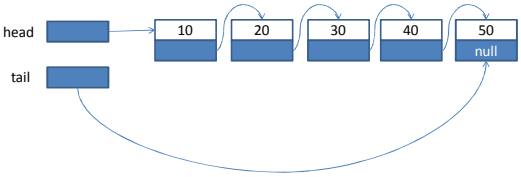
- add(10); After adding one element



62

Implement a Queue

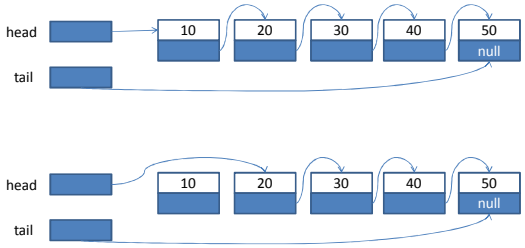
- After adding four more elements



63

Implement a Queue

- Remove first element



64

Implement a Queue

```
================Link.java======================
class Link{
    int value;
    Link next;
}
================MyQueue.java=====================
class MyQueue{
    private Link head, tail;
    public MyQueue(){
          head=tail=null;
    }
```

65

Implement a Queue

```
public void add(int value){
      Link l=new Link();
      l.value=value;
      if(head==null){
            head=tail=l;
      }else{
            tail.next=l;
            tail=l;
      }
}
```

66

Implement a Queue

```
public void display(){
    Link cur;
    cur=head;
    System.out.print("Queue Members: ");
    while(cur!=null){
        System.out.print(cur.value+" ");
        cur=cur.next;
    }
}
```

67

Implement a Queue

```
public int remove(){
    Link removedElement;
    if(head!=null){
        removedElement=head;
        head=head.next;
        return removedElement.value;
    }else{
        return -1;
    }
}
}
```

68

Implement a Queue

```
class TestQueue{
    public static void main(String[] args){
        MyQueue queue=new MyQueue();
        queue.add(10);
        queue.add(20);
        queue.add(30);
        queue.add(40);
        queue.add(50);
```

69

Implement a Queue

```
        queue.display();
        queue.remove();
        System.out.println("Removed Element:
"+queue.remove());
        queue.display();
        System.out.println("Removed Element:
"+queue.remove());
        queue.display();

    }
}
```

70

Implement a Queue

```
D:\programs\CJT\programs\classes and object>java TestQueue
Queue Members: 10 20 30 40 50 Removed Element: 20
Queue Members: 30 40 50 Removed Element: 30
Queue Members: 40 50
D:\programs\CJT\programs\classes and object>
```

71

Lecture-5

72

Package

- Package is a container for Java classes.
- Package allows to have qualified name for classes.
  - It allows class name collision.
- We can use same name for two java classes, if the classes are present in different packages.
- A java package organizes classes into namespaces.
- Classes of same category or providing similar functionalities should be placed in the same package.
- Packages can be stored in .jar file (Java Archive, compressed file). E.g., libraries are available as .jar files.

73

Package

- To create a package, we need to write package statement as a first statement in .java class.

  package mypackage;
- We can have hierarchical name for packages.

  package java.awt.image;
- Frequently, a package name begins with the top level domain name of the organization and then the organization's domain and then any subdomains listed in reverse order. E.g., in.ac.ddu
- If we use package, then .class files must be placed in the directory having the same name as package name.

74

Package Design Guidelines

- Only closely related classes should be placed in the same package.
- Classes that change together should be placed in the same package.
- Classes that are not reused together should not be placed in the same package.

75

Create package

- Suppose we have Test class in a package pack1.
- We create a directory having name pack1.

========= pack1\Test.java ======================

```
package pack1;
class Test{
    public static void main(String[] args){
        System.out.println("Hello");
    }
}
```

- Compile Test.java

  pack1 $ javac Test.java
- Now, Test class is part of the package: pack1.
- We can refer to that class using pack1.Test

76

Run java class that is part of package

- Go one level up in directory structure.
- Suppose pack1 directory is under test directory, make test directory as current working directory.

  D:\test\pack1
- Execute the Test class by referring it as pack1.Test

  test $ java pack1.Test

77

Set CLASSPATH

- But, if we want to access a Java class from any other directory, not the parent directory, i.e., pack1.
- We can set CLASSPATH for pack1
- CLASSPATH is environment variable used by Java to hold path of Java packages and jar files.
- On Windows, write the following as the value of CLASSPATH environment variable

  %CLASSPATH%;<path-parent-directory-of-pack1>

  %CLASSPATH%;D:\test;
- On Windows, we can set CLASSPATH using
  - My Computer -> Properties -> Advanced -> Environment Variables

78

Set CLASSPATH

- If classes are placed in a compressed java file (.jar), we can also set path of this jar file into CASSPATH
- Suppose our package(s) are in mylib.jar file
- CLASSPATH=%CLASSPATH%;D:\test\mylib.jar;

79

Use of import

- The import statement allows to use classes present in other packages.
- We can import the class Test using the following:
  import pack1.Test
- We can import all classes using a single statement
  import pack1.*;
- We should import only one class using one import statement
  - To avoid importing unnecessary classes and
  - When two packages contain a class with the same name? The class of the first import is considered.

80

Example: Use of import

- We have two packages: pack1 and pack2
- The pack1 package contains Box class
- The pack2 package contains BoxUse class that uses Box class

81

Example: Use of import

========== pack1\Box.java ==========

```
package pack1;
public class Box{
   int length;
   int width;
   int height;
   public String toString(){
        return "Box: "+length+"X"+width+"X"+height;

   }
}
```

82

Example: Use of import

========== pack2\BoxUse.java ==========

```
package pack2;
import pack1.Box;
class BoxUse{
   public static void main(String[] args){
        Box b=new Box();
        System.out.println(b);
   }
}
```

83

Exercise: Use of import

- Put the Stack class and its user class in different packages

84

Math class and its methods

- Most are static methods
  - Math.PI (double value)
  - Trigonometric methods
    - public static double sin(double a);
    - public static double cos(double a);
    - public static double tan(double a);
    - public static double asin(double a);
    - public static double acos(double a);
    - public static double atan(double a);

85

Math class and its methods

- Most are static methods
  - Exponent methods
    - public static double exp(double a);
    - public static double pow(double a, double b);
    - public static double log(double a);
    - public static double sqrt(double a);
  - Other useful methods:
    - min(), max, abs(), and random()

86

Private Constructor

- Constructor in Math is private. Cannot create its instance
- How to create an instance of a class that has private constructor
  - Create a static method and return an object from that.

```
public static Box getInstance(){
   return new Box(1,1,1);
}
```

87

Exercise Problems:

- Shuffle a deck of playing cards. Rank of Cards: 2, 3,..10, Jack, Queen, King, Ace. Suit of cards: Club, Diamond, Heart, Spade
- Implement a command line calculator that allows following operations: exp, pow, log, and sqrt. Usage is as follows: operation operands. E.g. the following performs exponential operation: java Calc e 10

88