This image cannot currently be displayed.

Chapter 6

Classes and objects

Declaration

- These slides are made for UIT, BU students only. I am not holding any copy write of it as I had collected these study materials from different books and websites etc. I have not mentioned those to avoid complexity.

Topics

- Class
  - The General form of a class
  - Difference with C++
  - A Simple Class
- Object
  - Declaring Object
  - Assigning object reference variables
- Constructors
- this keyword
- Garbage Collection
- The finalize() method

Physical Object

- An object is any physical or conceptual thing that we find around us.
  - Examples: classmates, table, television, stereo etc.
- Characteristics : identity, state and behavior
  - Identity is the name given to an object, like name given to a dog.
  - A dog has state [name, colour, breed] and
  - behavior [barking, fetching, and wagging tail]

Object

- In Object Oriented Programming we are trying to model either real-world entities or processes and represent them in software.
- There are compelling reasons for modeling entities
  - A model is simplification of reality. We model because we cannot comprehend the complexity of a system in its entirely.
  - We model to visualize, specify, construct, and document the structure and behavior of a system’s architecture.
  - A model is a complete description of a system from a particular perspective.

Object

- Software objects are modeled after real-world objects such that they too have state and behavior.
- A Software object maintains its state in one or more variables. A variable is some data named by an identifier.
- A software object implements its behavior with methods. A method is a function (subroutine) associated with an object.
- An object is a software bundle of variables and related methods.
- Real world objects can be represented using software objects. You can represent a dog as a software object in an animation program or a stereo as a software object in the program that plays music.

Object

- Example: Person Object
- Attributes
  1. First Name
  2. Last Name
  3. Age
  4. Weight
- Methods
  1. set_first_name()
  2. get_first_name()
  3. set_last_name()
  4. get_last_name() etc.

Message

- In a software application an object usually appears as a component of a larger program that contains many other objects. Through the interaction of these objects programmers achieve functionality of the application.
- Software objects interact and communicate with each other by sending messages to each other.
- When object A wants object B to perform one of B’s methods, object A send message to object B.

Message

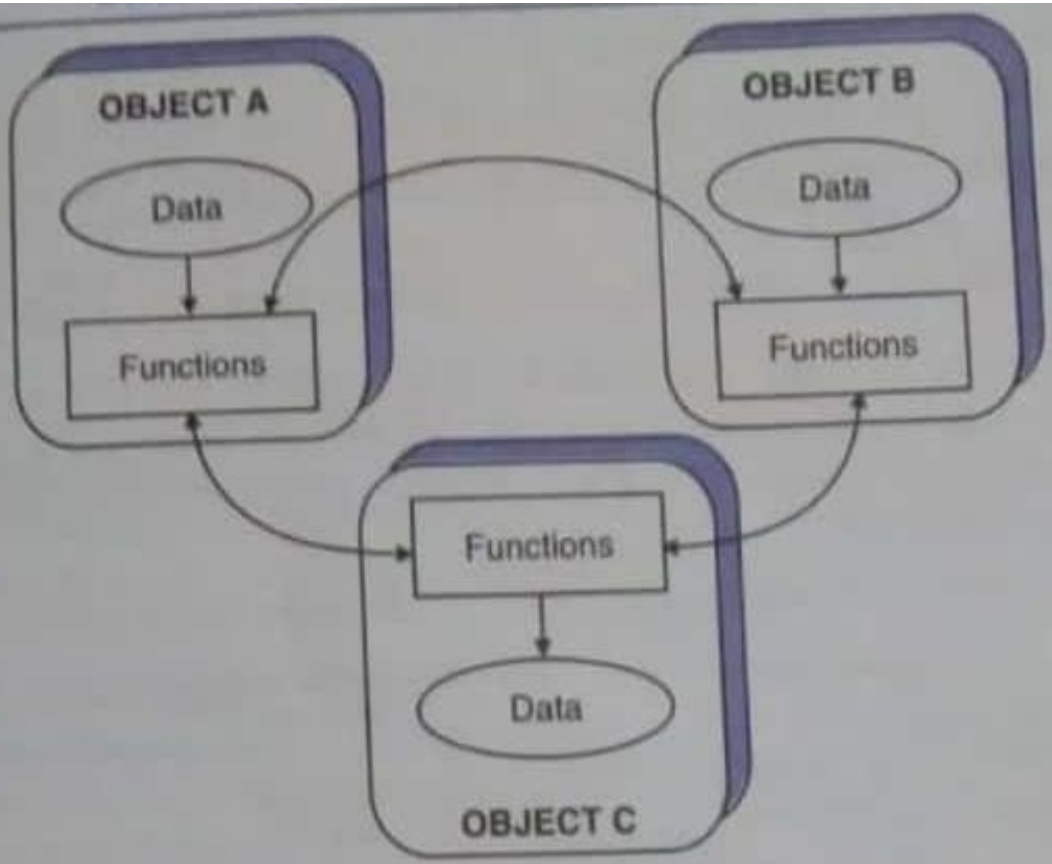
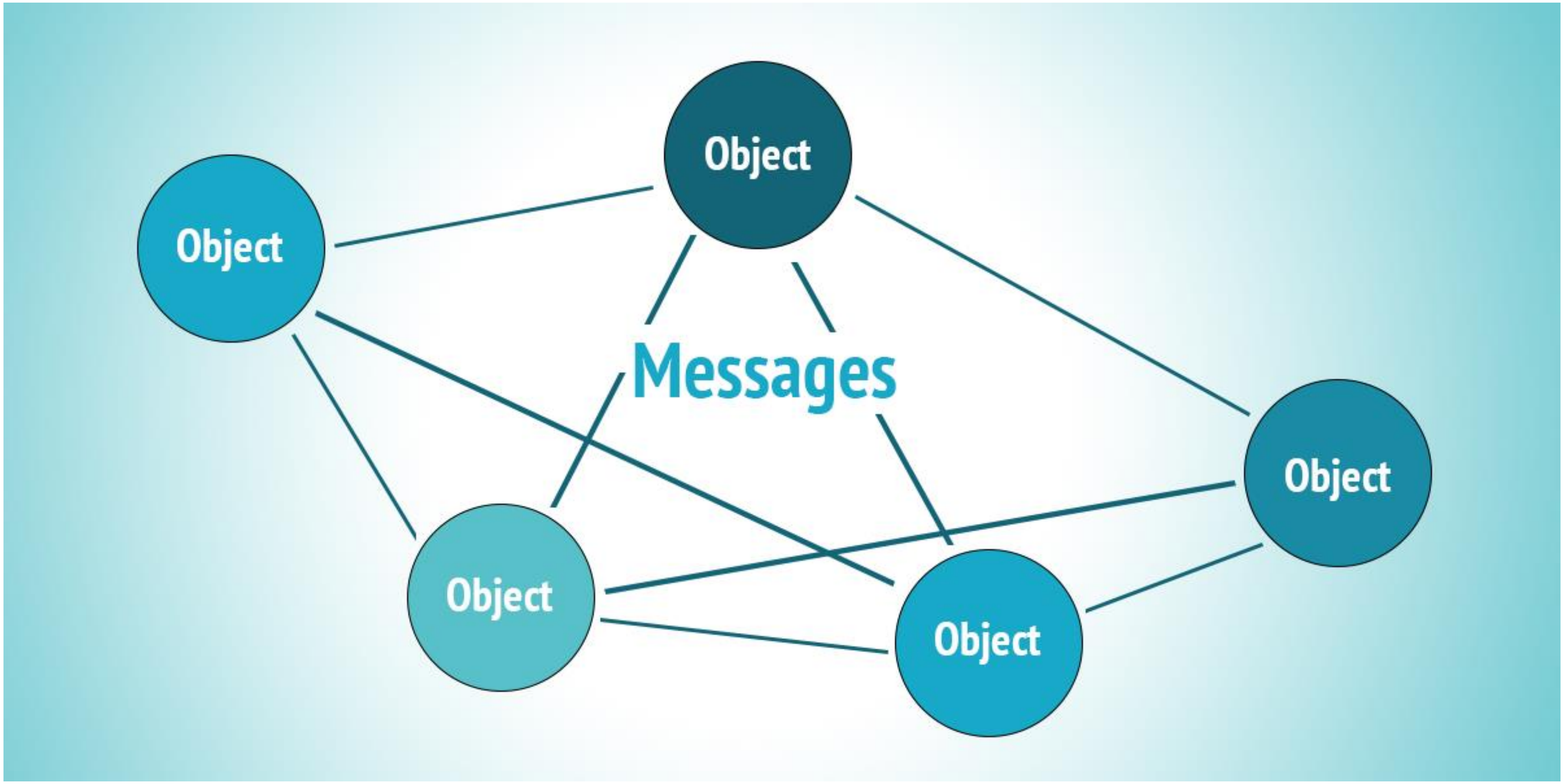


Figure 3.3: During program execution one object passes and receives information from another object using member functions. Such process is called Data messaging.

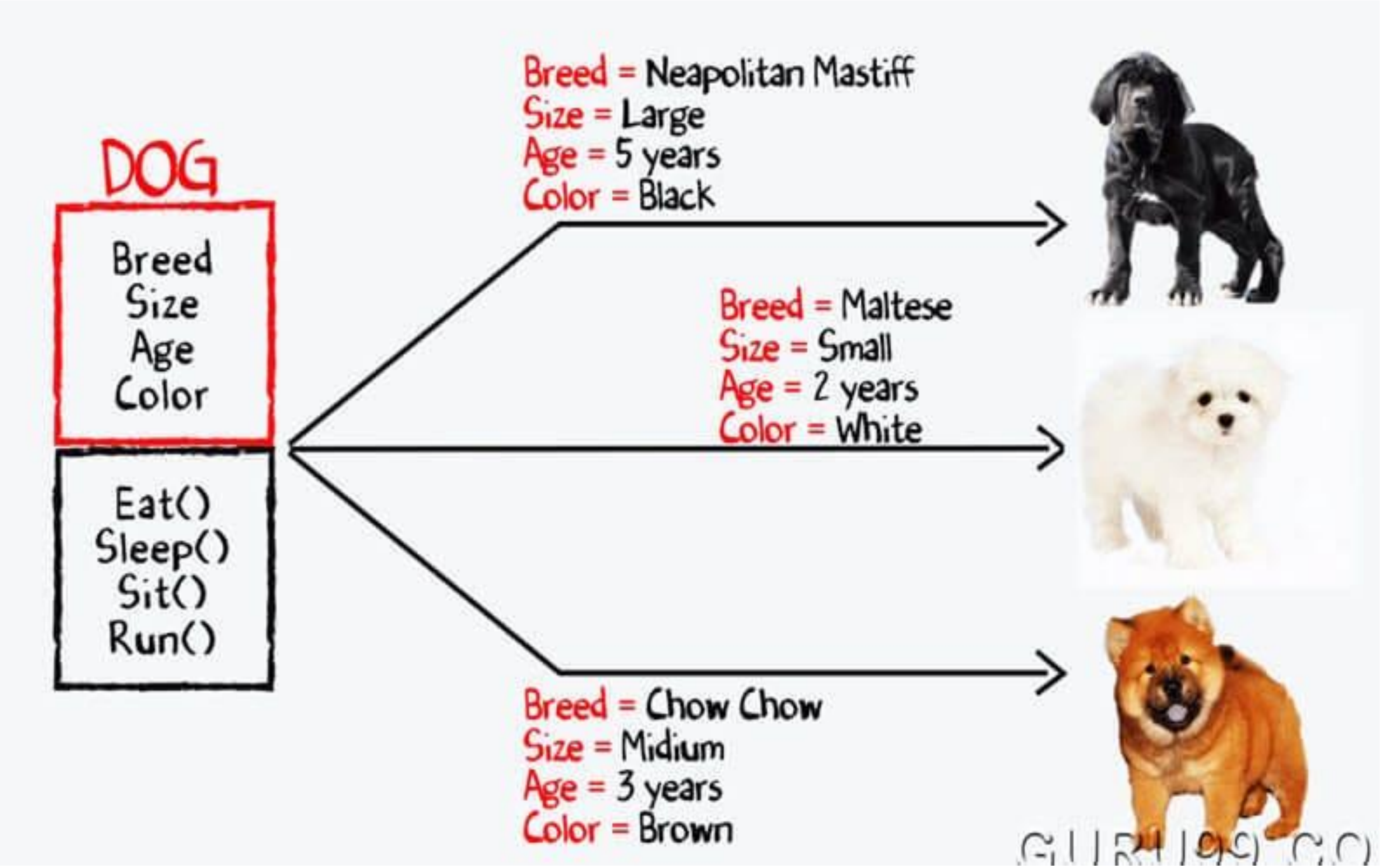
Message



Class

- A class is a blueprint or prototype, that defines the variables and the methods common to all objects of a certain kind.
- Example: Television
  - Instance Variables necessary to contain the current channel, the current volume etc. for the television object.
  - Instance methods that allow the viewer to change channels, volume etc.
- After creating the television class, any number of television objects from the class can be created. When a instance of a class is created, the system allocates enough memory to the object and all its instance variables. Each instance gets it's own copy of all the instance variables in the class.

Class and Objects



DOG

Breed
Size
Age
Color

Eat()
Sleep()
Sit()
Run()

Breed = Neapolitan Mastiff
Size = Large
Age = 5 years
Color = Black

Breed = Maltese
Size = Small
Age = 2 years
Color = White

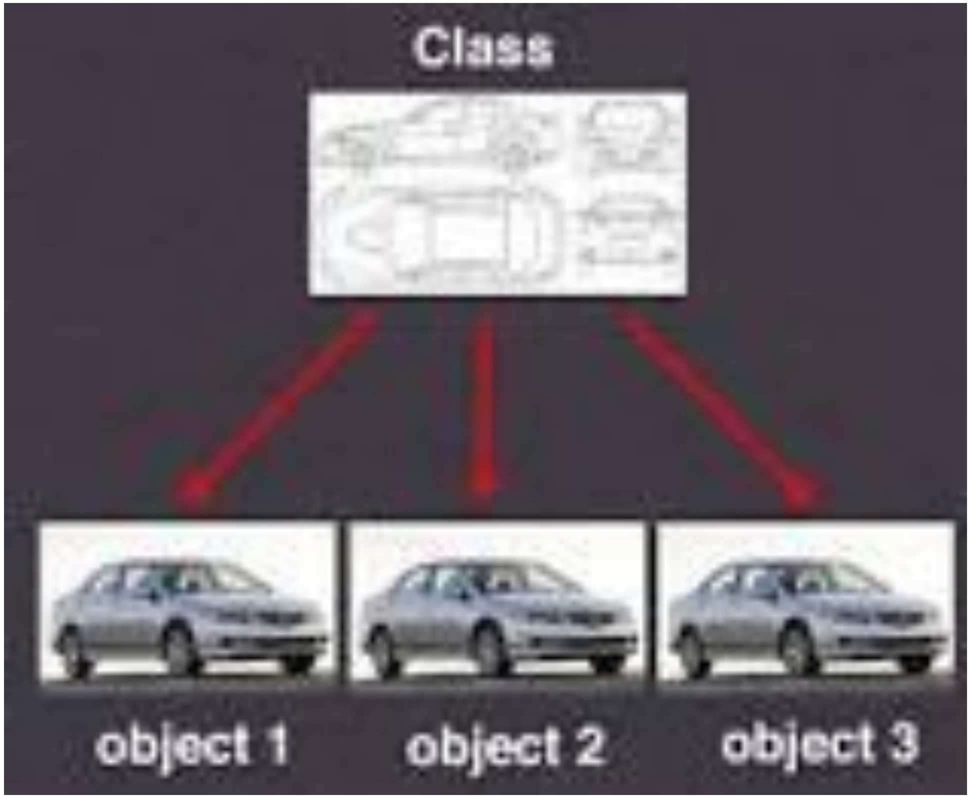
Breed = Chow Chow
Size = Midium
Age = 3 years
Color = Brown

Class

- In addition to instance methods and instance variables, classes can define following.
- Class Variables (Static Variable), which contain information that is shared by all instances of the class.
  - For example, suppose that all television had the same number of channels. In this case, you can define a class variable that contains the number of channels. All instance share this variable. If one object changes the variable, it changes for all other objects of that type.
- Class Methods (Static Method), which can be invoked directly from the class, whereas instance methods are invoked on a particular instance.

Class vs Object

- The difference is simple and conceptual. A **class** is a template for **objects**. ... An **object** is a member or an "instance" of a **class**.



Class vs Object

- A class is not itself the object it describes; a blueprint of a car is not a car.
- Objects are entities that actually exist in computer software terms, objects will exist in the memory in the computer, whereas class is not.

Class vs Object

No.	Object	Class
1)	Object is an **instance** of a class.	Class is a **blueprint or template** from which objects are created.
2)	Object is a **real world entity** such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.	Class is a **group of similar objects**.
3)	Object is a **physical** entity.	Class is a **logical** entity.
4)	Object is created through **new keyword** mainly e.g. Student s1=new Student();	Class is declared using **class keyword** e.g. class Student{}
5)	Object is created **many times** as per requirement.	Class is declared **once**.
6)	Object **allocates memory when it is created**.	Class **doesn't allocated memory when it is created**.
7)	There are **many ways to create object** in java such as new keyword, newInstance() method, clone() method, factory method and deserialization.	There is only **one way to define class** in java using class.

Class

- The Class is the core of Java.
- It defines new data types.
- Once defined, this new type can be used to create objects of that type.

The General form of a class

```
Class classname{
        type instance_variable1;
        type instance_variable2;
        //---
        type instance_variableN;
        type methodname1(parameter_list){
             //body of method
        }
        type methodname2(parameter_list){
             //body of method
        }
        //---
        type methodnameM(parameter_list){
             //body of method
        }
}
```

Difference with C++

- Unlike C++, specification, declaration and implementation of methods in one place makes the code big, but in the long run it is easier to maintain.

A Simple Class

```
class Box {
  double width;
  double height;
  double depth;
}
```

- class declaration only creates a template.
- It does not create actual object.
- Thus the preceding code does not cause any objects of type Box to come into existence.

Object

Box mybox = new Box();

mybox will be an instance of Box.

- An object contains its own copy of each instance variable defined by the class.
- To access these variables, you will use the dot(.) operator.

mybox.width =10.0;

Example

```
class Box {
  double width;
  double height;
  double depth;
}

// This class declares an object of type Box.
class BoxDemo {
  public static void main(String args[]) {
    Box mybox = new Box();
    double vol;
```

Example

```
// assign values to mybox's instance variables
    mybox.width = 10;
    mybox.height = 20;
    mybox.depth = 15;

    // compute volume of box
    vol = mybox.width * mybox.height * mybox.depth;

    System.out.println("Volume is " + vol);
  }
}
```

Another Example

```
// This program declares two Box objects.
class Box {
  double width;
  double height;
  double depth;
}

class BoxDemo2 {
  public static void main(String args[]) {
    Box mybox1 = new Box();
    Box mybox2 = new Box();
    double vol;
```

Another Example

```
// assign values to mybox1's instance variables
   mybox1.width = 10;
   mybox1.height = 20;
   mybox1.depth = 15;

   /* assign different values to mybox2's
     instance variables */
   mybox2.width = 3;
   mybox2.height = 6;
   mybox2.depth = 9;
```

Another Example

```
// compute volume of first box
  vol = mybox1.width * mybox1.height * mybox1.depth;
  System.out.println("Volume is " + vol);

  // compute volume of second box
  vol = mybox2.width * mybox2.height * mybox2.depth;
  System.out.println("Volume is " + vol);
 }
}
```

Adding method

```
// This program includes a method inside the box class.
class Box {
  double width;
  double height;
  double depth;

  // display volume of a box
  void volume() {
    System.out.print("Volume is ");
    System.out.println(width * height * depth);
  }
}
```

Adding method

```
class BoxDemo3 {
 public static void main(String args[]) {
  Box mybox1 = new Box();
  Box mybox2 = new Box();

  // assign values to mybox1's instance variables
  mybox1.width = 10;
  mybox1.height = 20;
  mybox1.depth = 15;
```

Adding method

```
/* assign different values to mybox2's
    instance variables */
  mybox2.width = 3;
  mybox2.height = 6;
  mybox2.depth = 9;

  // display volume of first box
  mybox1.volume();

  // display volume of second box
  mybox2.volume();
 }
}
```

Declaring Object

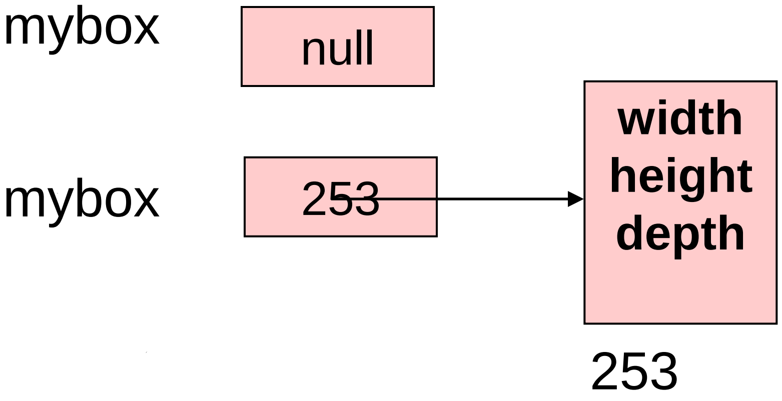
Box mybox = new Box();

It can be rewritten by

Box mybox;//declare reference to object

mybox = new Box(); //allocate a Box object

- Effect



Reference vs Pointer

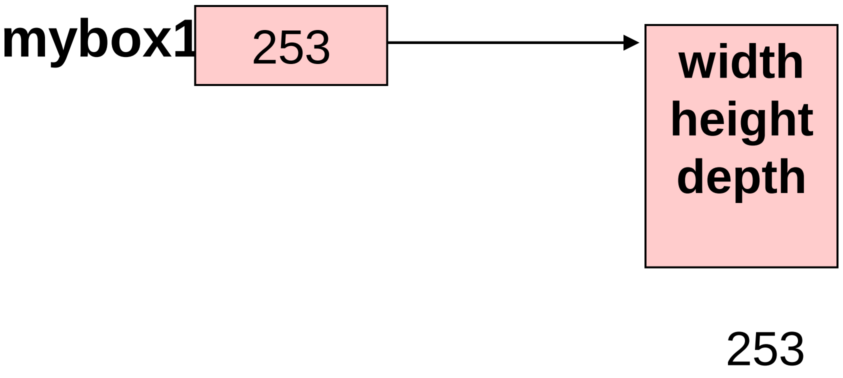
- Here reference appear to be similar to memory pointer.
- The main difference with C++; In Java you cannot manipulate references as you can in case actual pointer.
- Thus you cannot cause an object reference to point to an arbitrary memory location or manipulate it like an integer.

Java primitive type and user defined type

- **new** operator is not required when we are using integer or character.
- Java’s simple types are not implemented as objects.
- They are implemented as normal variable.
- This is done in the interest of efficiency.
- **new** allocates memory for an object during runtime.
- In case of shortage of memory, **new** method cannot allocate memory and a run-time exception will occur.

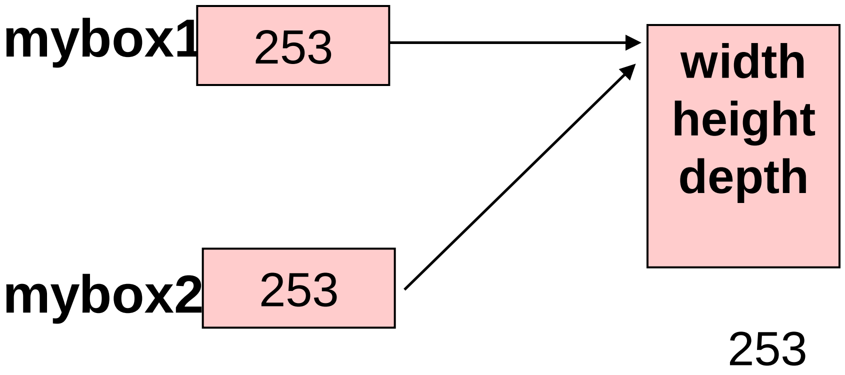
Assigning object reference variables

Box mybox1 = new Box();



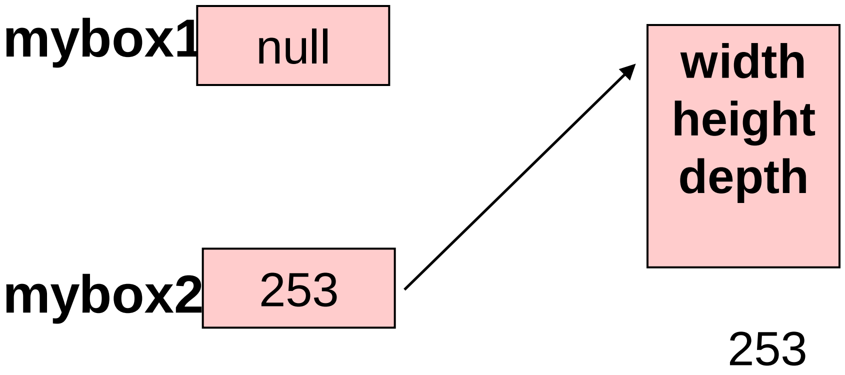
Assigning object reference variables

Box mybox2 = mybox1;



Assigning object reference variables

mybox1 = null;



Constructors

mybox = new Box();

- The class name followed by parentheses specifies the constructor of the class.
- If no explicit constructor is specified, then java will automatically supply a default constructor.
- A constructor initializes an object immediately upon creation.
- It has no return type, not even void.
- Implicit return type of a class constructor is the class type itself.

Constructors

```
/* Here, Box uses a constructor to initialize the dimensions of a box.*/
class Box {
 double width;
 double height;
 double depth;
Box() {// This is the constructor for Box.
  System.out.println("Constructing Box");
  width = 10;
  height = 10;
  depth = 10;
 }
double volume() {// compute and return volume
  return width * height * depth;
 }
}
```

this keyword

- Sometimes a method will need to refer to the object that invoked it.
- this can be used inside any method to refer to the current object.

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

Garbage Collection

- Objects are dynamically allocated memory by using the **new** operator.
- In C++, dynamically allocated memory must be manually released by use of a **delete** operator.
- In java deallocations is done automatically by a technique called garbage collection.
- When no references to an object exist, that object is assumed to be no longer needed and the memory occupied by the object can be reclaimed.
- There is no explicit need to destroy objects as in C++.
- Garbage collection only occurs sporadically (if at all) during the execution of your program.
- It will not occur simply because one or more objects exist that are no longer used.

The finalize() method

- Sometimes an object will need to perform some action when it is destroyed.
- If an object is holding some non-java resource such as a file handle or window character font, then you might want to make sure these resources are freed before an object is destroyed.
- To handle such situation, Java provides a mechanism called finalize().
- By using finalize() you can define specific actions that will occur when an object is just about to be reclaimed by the garbage collector.

```
protected  void finalize() {
 //finalization code here
}
```

The finalize() method

- You cannot know when finalize() will be executed.
- You must not rely on finalize() for normal program execution.
- Java does not support destructor as in C++.
- The finalize() method only approximates the function of destructor.

Why is the finalize () method in java.lang.Object protected?

- The finalize method is intended to be executed by the JVM just before Garbage Collection collects garbage.
- What is **the minimum accessibility you could give to a method so that it could be overridden by any other subclass of object**?
- The answer is: *protected*.
- WHY?
- If it had been marked as ***private*** then it could not be accessed out side ***java.lang.Object*** class. And because of it you cannot inherit it at all. So, in turn you cannot override it.
- If it had default access level i.e. ***package level access*** then it could not be accessed the world outside ***java.lang*** **package**. And because of it you cannot inherit it to a class outside java.lang package. So, in turn you cannot override it in your own class.

End of Chapter 6

Questions?