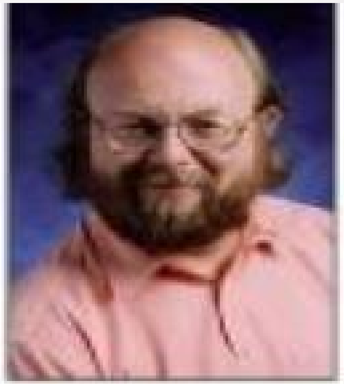
# Lumen -Campus -2021

Day -1



# OOP with Java

# K.Srivatsan - vatsank@gmail.com

- IT Trainer Since 1991
- Conducting Java Training Since 2004
- About 60+ Corporate Clients

| | | |
|---|---|---|
| CSC | IBM | accenture |
| verizon | amdocs | 3i Infotech<br>Innovation • Insight • Integrity |
| Standard Chartered<br>Scope International | ERICSSON | TMB<br>Tamilnad Mercantile Bank Ltd |
| HCL | Hexaware<br>TECHNOLOGIES | Sapient |
| | | |

# History of Java

- Conceived as Programming language for embedded systems like microwave ovens, televisions etc
- One of the first projects developed using Java
  - personal hand-held remote control named Star 7.
- The original language was called Oak
- It was developed in the year 1991 at Sun Microsystems

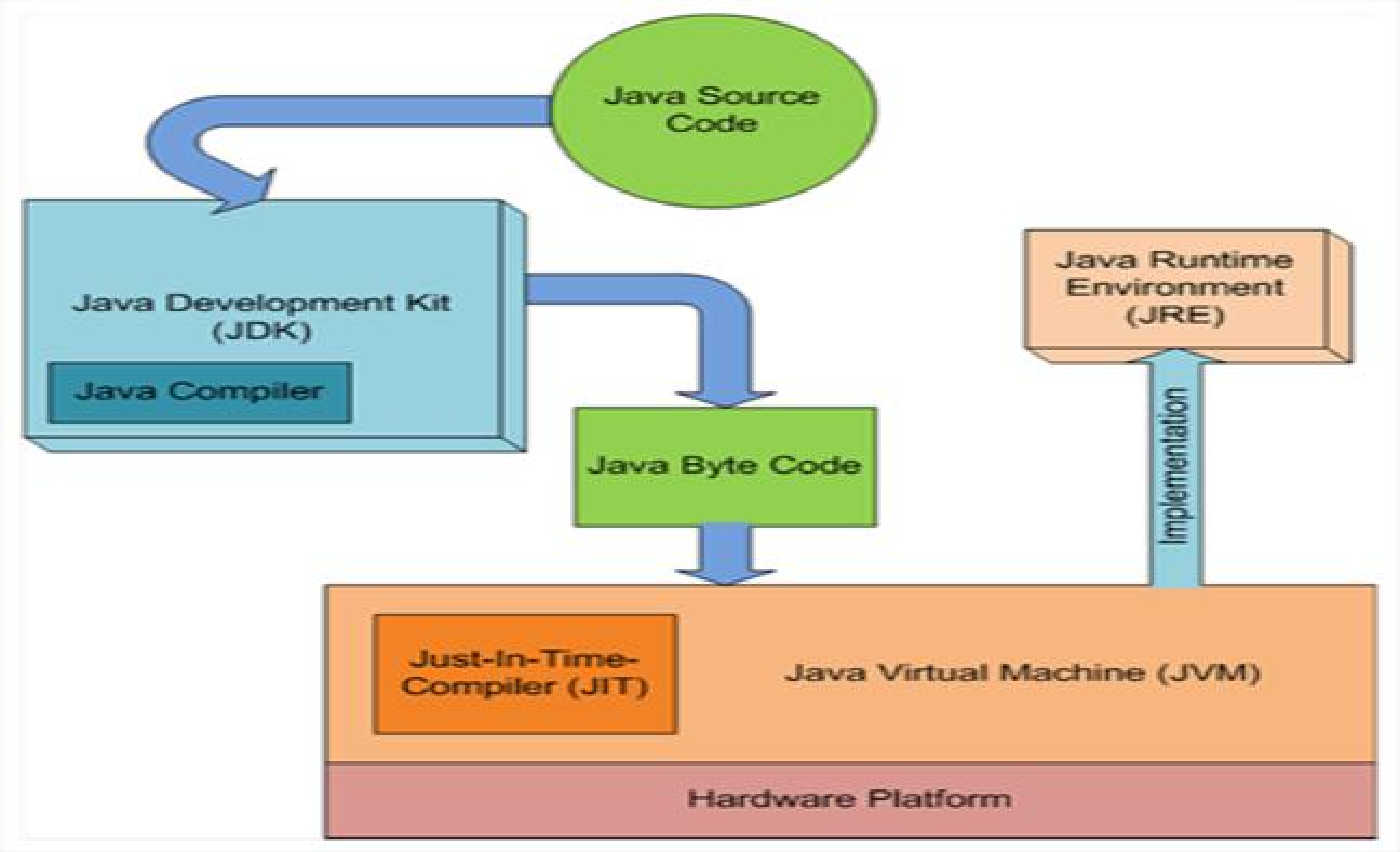
# Primary Goals of Java

- **Provides an easy-to-use language by:**
  - Avoiding many pitfalls of other languages
  - Being object-oriented
  - Enabling users to create streamlined and clear code
- **Provides an interpreted environment for**:
  - Improved speed of development
  - Code portability
  - Enables users to run more than one thread of activity
  - Loads classes dynamically; that is, at the time they are actually needed
  - Supports changing programs dynamically during runtime by loading classes from disparate sources

# The Java Virtual Machine

- Executes instructions that a Java compiler generates.
- A runtime environment, embedded in web browsers, servers, and operating systems
- Imaginary machine that is implemented by emulating software on a real machine
- Reads compiled byte codes that are platform-independent
- **Bytecode**
  - a special machine language that can be understood by the JVM independent of any particular computer hardware,
  - **Java Runtime Environment (JRE) is an implementation of the JVM".**
  - **"JVM becomes an instance of JRE at runtime".**
    - A runtime instance of JVM will born when **.class** file starts its execution

# JDK,JVM and JRE

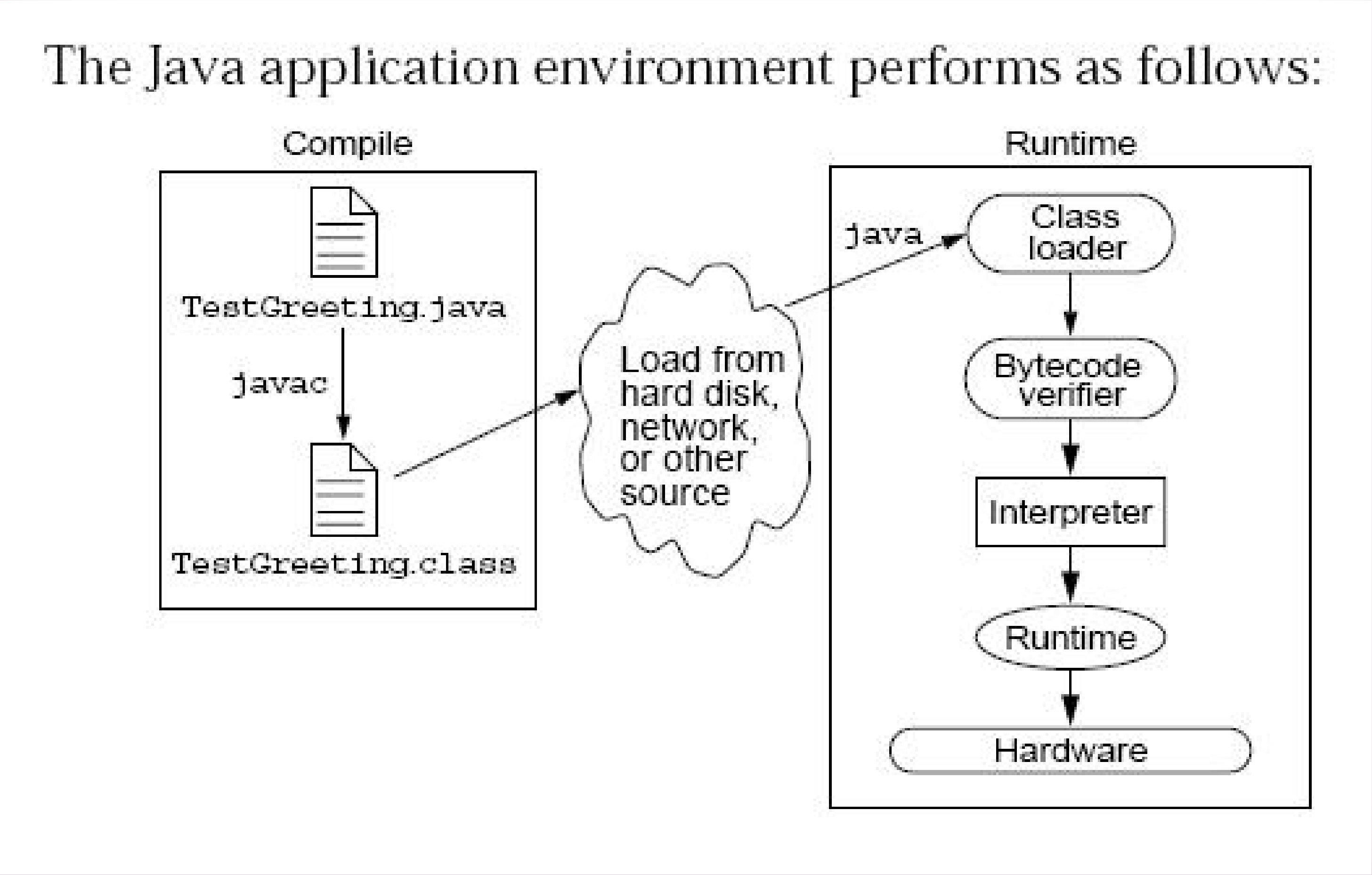

Java Source Code
Java Development Kit (JDK)
Java Compiler
Java Byte Code
Java Runtime Environment (JRE)
Implementation
Just-In-Time-Compiler (JIT)
Java Virtual Machine (JVM)
Hardware Platform

# JVM Tasks

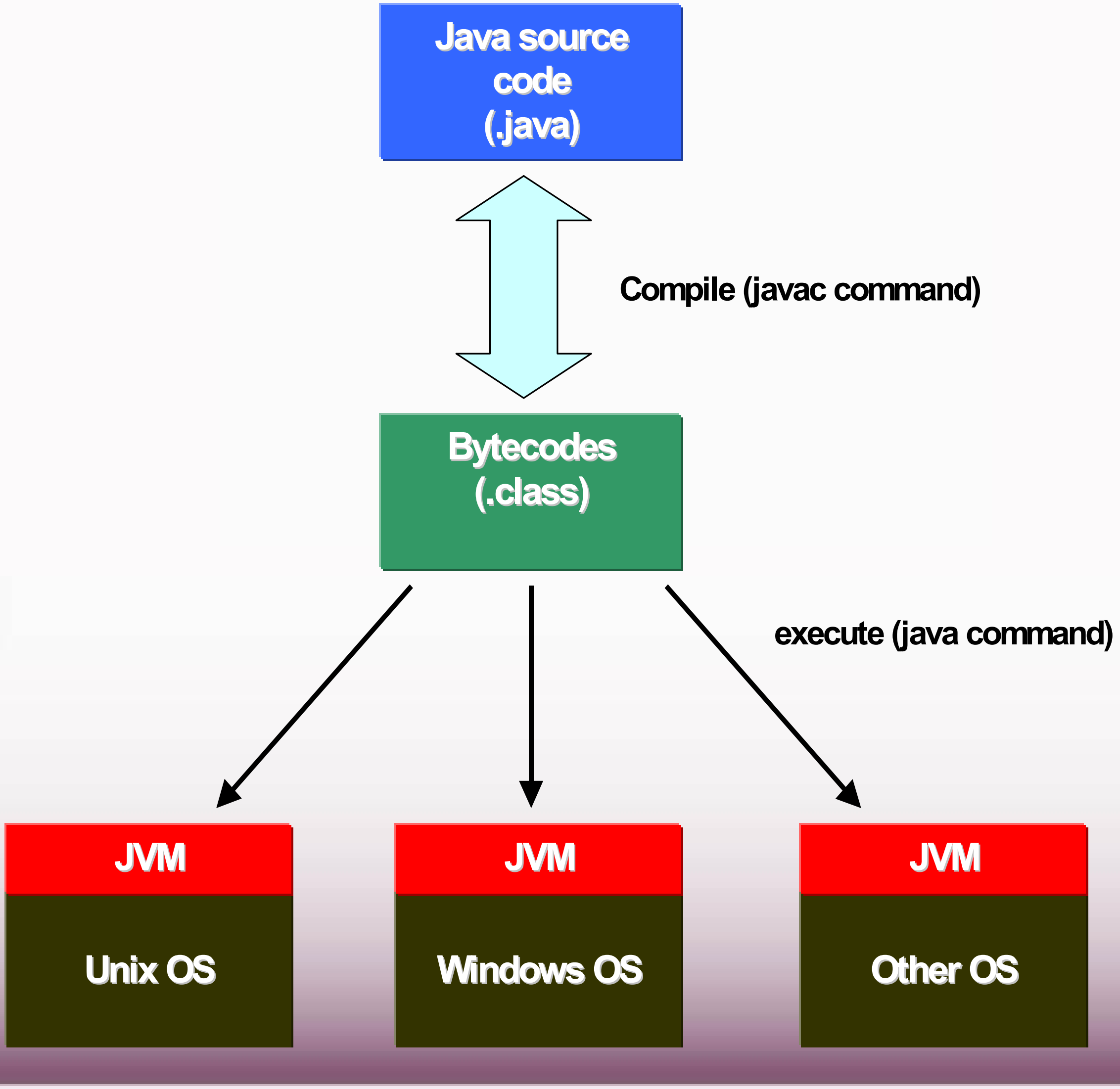
1. Loads code
   - Loads all classes necessary for the execution of a program
   - Maintains classes of the local file system in separate *namespaces*

2. Verifies code (Bytecode Verifier)
   - The code adheres to the JVM specification.
   - The code does not violate system integrity.
   - The parameter types for all operational code are correct.
   - No illegal data conversions have occurred.

# Java Run Time Environment



The Java application environment performs as follows:



Java source code (.java)

Compile (javac command)

Bytecodes (.class)

execute (java command)

JVM

Unix OS

JVM

Windows OS

JVM

Other OS

# Integrated Development Environments

- An IDE is the high-productivity tool
- Used to edit, compile, and test programs, manage projects, debugging, building GUI interfaces, etc.
- IDE provides extensive programming support for editing and  project management,
- The Popular IDE’s
  - Eclipse
  - NetBeans
  - JDeveloper

# Java language syntax

# Objectives

- Identify the basic parts of a Java program
- Differentiate among Java literals, primitive data types,
- variable types ,identifiers and operators
- Develop a simple valid Java program using the concepts learned in this chapter

# Program structure

- A program in Java consists of one or more class definitions
- One of these classes must define a method *main(),* which is where the program starts running

- Java programs should always end with the .java extension.
- There can be More than one Class Definition in a class ,but only one public class
- Source Filenames should match the name of the  public class.

# Source File Layout

- Basic syntax of a Java source file is:
- ***[<package_declaration>]***
- ***<import_declaration>**** 
- ***<class_declaration>+***

# Software Packages

- Packages help manage large software systems.
- Packages can contain classes and sub-packages.
- **package <*top_pkg_name*>[.<*sub_pkg_name*>]*;**
- Specify the package declaration at the beginning of the source file.
- Only one package declaration per source file.
- If no package is declared, then the class is placed into the default package.
- Package names must be hierarchical and separated by dots.

# The import Statement

- **import <*pkg_name*>[.<*sub_pkg_name*>]*.<*class_name*>;**

- **import <*pkg_name*>[.<*sub_pkg_name*>]*.*;**

- **import java.util.List;**
- **import java.io.*;**
- **import shipping.gui.reportscreens.*;**

- The import statement does the following:
  - Precedes all class declarations
  - Tells the compiler where to find classes

# Example 1.1

```
package com.training;

public class Greeting {


    public String getMessage() {

       return "Welcome to Java  Programming";

    }

}
```

# Example 1.1 (contd)

```
package com.training;

public class TestGreetings {

    public static void main(String[] args) {

       Greetings grtObj = new Greetings();


       System.out.println(grtObj.getMessage());

    }

}
```

# The System Class

- Its part of the java.lang package
- The classes in this package are available without the need for an import statement
- This  class contains several useful class fields and methods.
- It can’t be Instantiated
- It also  Provides facilities for
  - Standard Input
  - Standard Output
  - Error Output Streams
  - Access to externally defined properties

# Declaring Java Technology Classes

- Basic syntax of a Java class:

```
<modifier>* class <class_name> {
<attribute_declaration>*
<constructor_declaration>*
<method_declaration>*
}
```

- Define an attribute:

- `<modifier>* <type> <name> [ = <initial_value>];`

# Declaring Methods

- Basic syntax of a method:
- ***<modifier>* <return_type> <name> ( <argument>* ) {***
- ***<statement>****
- ***}***

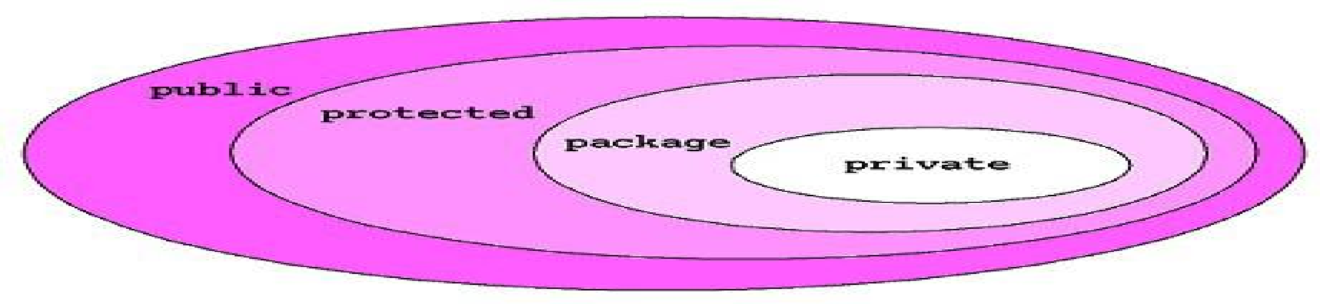
```
public int getWeight() {
return weight;
}

public void setWeight(int newWeight) {
        if ( newWeight > 0 ) {
             weight = newWeight;
        }
     }
```

# Accessing Object Members

- The *dot notation is:* ***<object>.<member>***
  - used to access object members, including attributes and methods.

- d.setWeight(42);
- d.weight = 42; // only permissible if weight is public

# The 4 Access Levels and 3 Modifiers

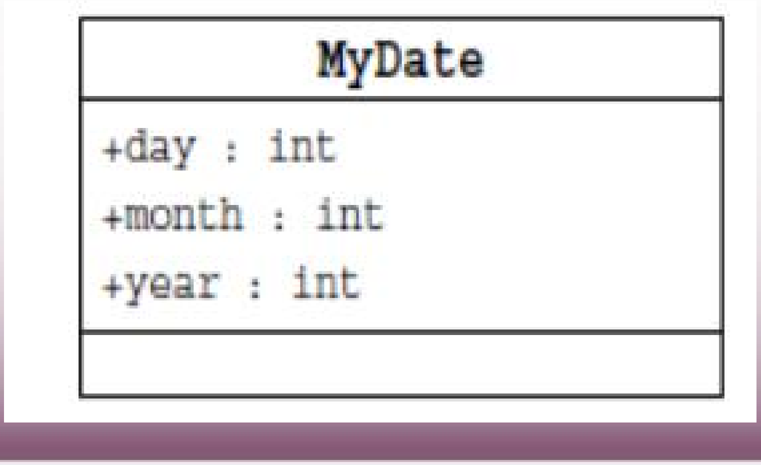


| | Private | No modifier | Protected | Public |
|---|---|---|---|---|
| Same class | Yes | Yes | Yes | Yes |
| Same package - subclass | No | Yes | Yes | Yes |
| Same package - nonsubclass | No | Yes | Yes | Yes |
| Different package - subclass | No | No | Yes | Yes |
| Different package - nonsubclass | No | No | No | Yes |

# Information Hiding

- Client code has direct access to internal data
- (d refers to a MyDate object):

- d.day = 32; // invalid day
- d.month = 2; d.day = 30; // plausible but wrong
- d.day = d.day + 1; // no check for wrap around

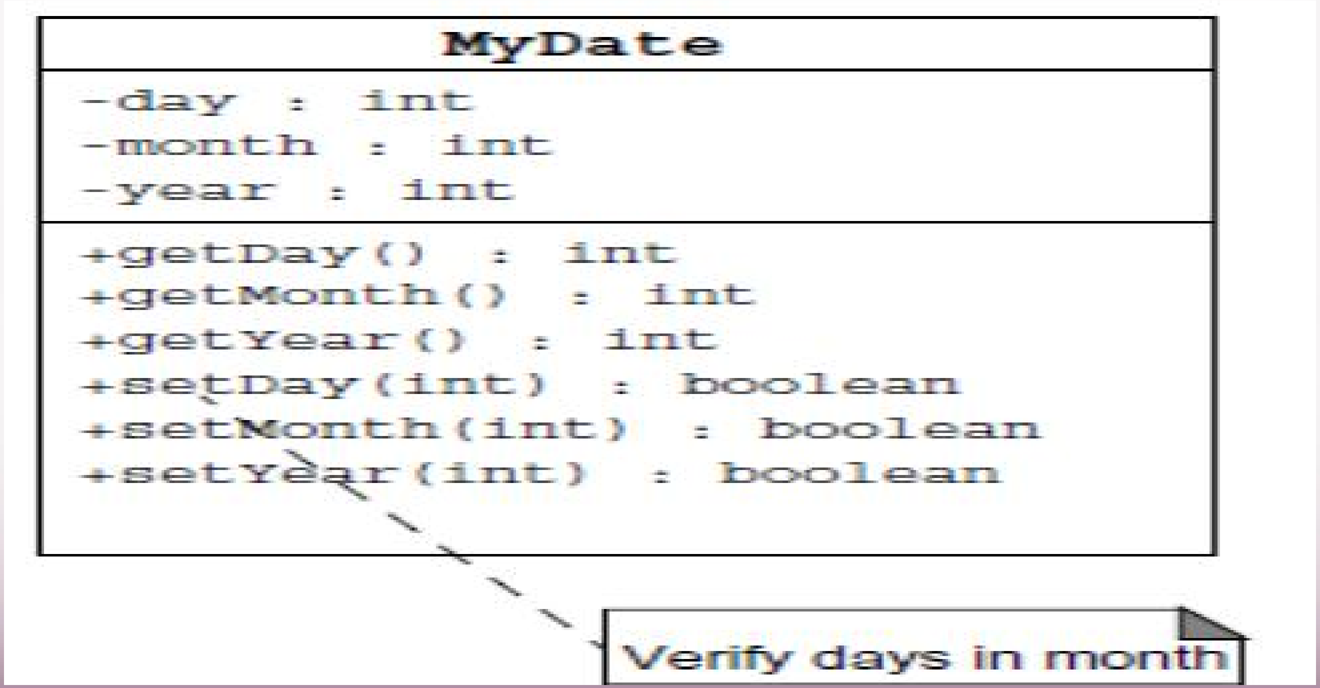


MyDate

+day : int
+month : int
+year : int

# Encapsulation

- Hides the implementation details of a class
- Forces the user to use an interface to access data
- Makes the code more maintainable



```
MyDate
-day : int
-month : int
-year : int
+getDay() : int
+getMonth() : int
+getYear() : int
+setDay(int) : boolean
+setMonth(int) : boolean
+setYear(int) : boolean
```

Verify days in month

# Classes

Top-level classes can be declared as

- public
  - a public class is globally accessible.
  - a single source file can have only *one* public class or interface
- abstract
  - an abstract class cannot be instantiated
- final
  - a final class cannot be subclassed
- Default
  - With any Modifier

- They can’t be declared as protected and private

# Constructors

- Have no return type
- Have the same name as the class
- If we don't' put a constructor the compiler puts a default one
  - The default constructor has the *same access modifier as the class*.
  - The default constructor has *no arguments*.
  - The default constructor is *always* a no-arg constructor, but a no-arg constructor is not necessarily the *default* constructor
  - The default constructor includes a *no-arg call to the super constructor* (super()).
- They are not inherited and hence they are not overridden
- It can be Overloaded
- It can have any of  the Four Access Modifies
- It cannot be synchronized
- It can have throws clause

# Instantiation with new

- It is the process by which individual objects are created.
  - **Class objectReference = new Constructor();**
- Declaration
  - Employee empObj;
- Instantiation
  - `empObj = new Employee()`
- Declaration and Instantiation
  - `Employee empObj = new Employee()`
  - **new** operator allocates memory for an object.

# Constructor Overloading

- One constructor can call another overloaded constructor of the same class by using ***this*** keyword.
- this() is used to call a constructor from another overloaded constructor in the same class
- The call to this() can be used only in a constructor ,and must be the first statement in a constructor
- A constructor can call its super class constructor by using the ***super*** keyword.
- A constructor can have a call to **super() or this() but never both**

# Overloaded Constructor

Time.java

```
class Time
{
  private int hour,min,sec;

  // Constructor
  Time()
  {
   hour = 0;
   min = 0;
   sec = 0;
  }

  //Overloaded constructor
  Time(int h, int m, int s)
  {
    hour = h;
    min = m;
    sec = s;
  }
}
  // Code continues …
```

# *this* keyword

- Is a reference to the object from which the method was invoked

*this* keyword

```
Time(int hour, int min, int sec)
  {
     this.hour = hour;
    this.min = min;
    this.sec = sec;
  }
```

# Comments

- Java supports three forms of comments
  - `// single line comments`

  -
    ```
    /* multi
       line
       comment
       */
    ```

  -
    ```
    /** a
       * Javadoc
       * comment
       */
    ```

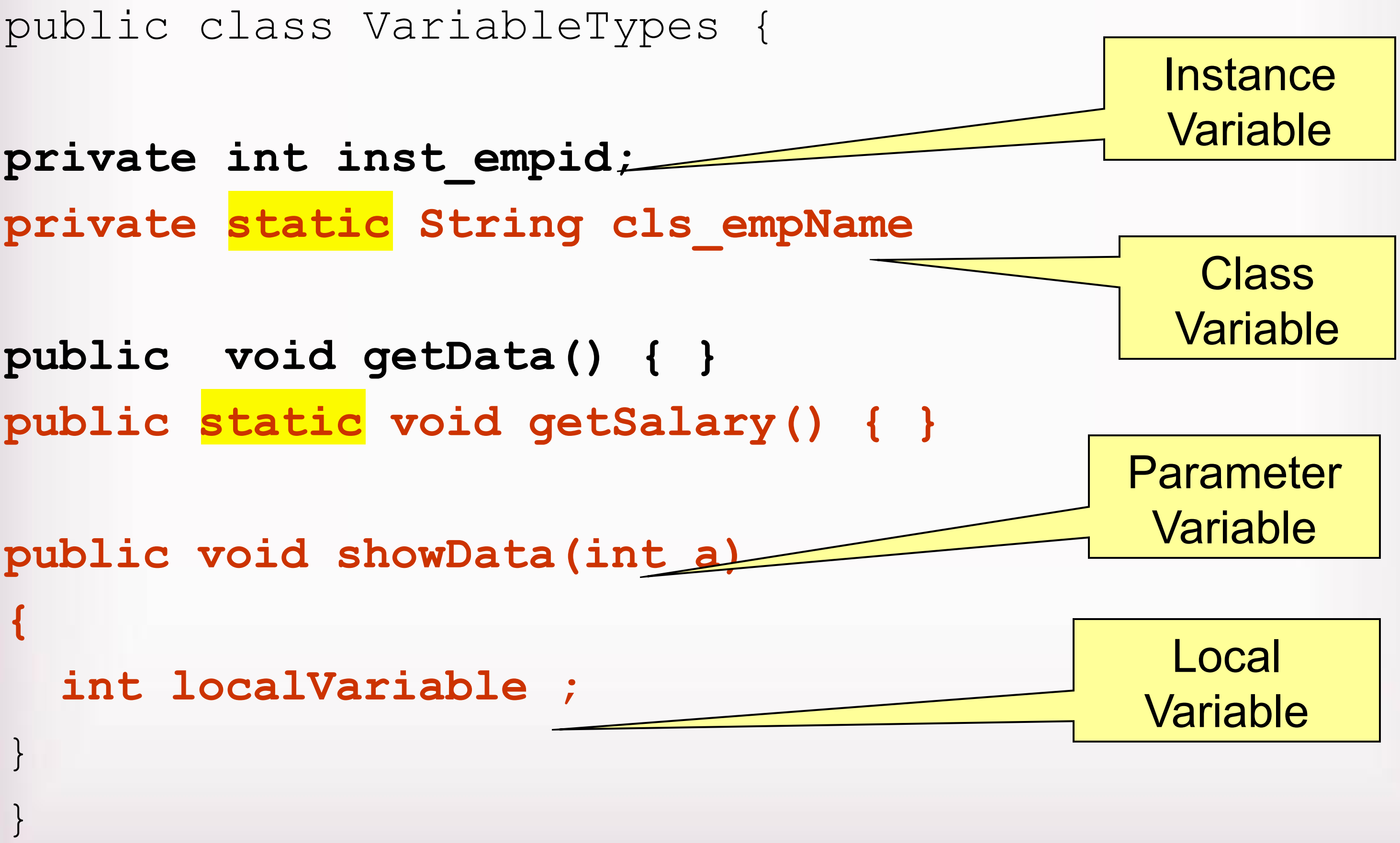
# Variables and Methods

- A **variable**, which corresponds to an attribute, is a named memory location that can store a certain type of value.
- Variable is a kind of special container that can only hold objects of a certain type.
- Primitive type Variable
  - Basic, built-in types that are part of the Java language
  - Two basic categories
    - boolean
    - Numeric
      - » Integral - byte, short, int, long, char
      - » Floating point - float, double

# Instance Variables and Methods

- Variables and methods can be associated either with objects or their classes
- An **instance variable and instance method** belongs to an object.
- They can have any one of the four access levels
  - Three access modifies – private, public , protected
- They have a default value
- **A class variable** (or **class method**) is a variable (or method) that is associated with the class itself.

# Example for Variables



```
public class VariableTypes {

private int inst_empid;
private static String cls_empName

public  void getData() { }
public static void getSalary() { }

public void showData(int a)
{
   int localVariable ;
}
}
```

Instance Variable

Class Variable

Parameter Variable

Local Variable

# Identifiers

- Identifiers are used to name **variables, methods, classes and interfaces**

- Identifiers
  - start with an alphabetic character
  - can contain letters, digits, or "_"
  - are unlimited in length

- Examples
  - *answer, total, last_total, relativePoint, gridElement, person, place, stack, queue*

# Local Variable needs to Be Initialized

```
public class LocalVariable {
   private String name;
public void display()
{
 int age;
_System.out.println("Age"+age);
 System.out.println("Name"+name);


}
```

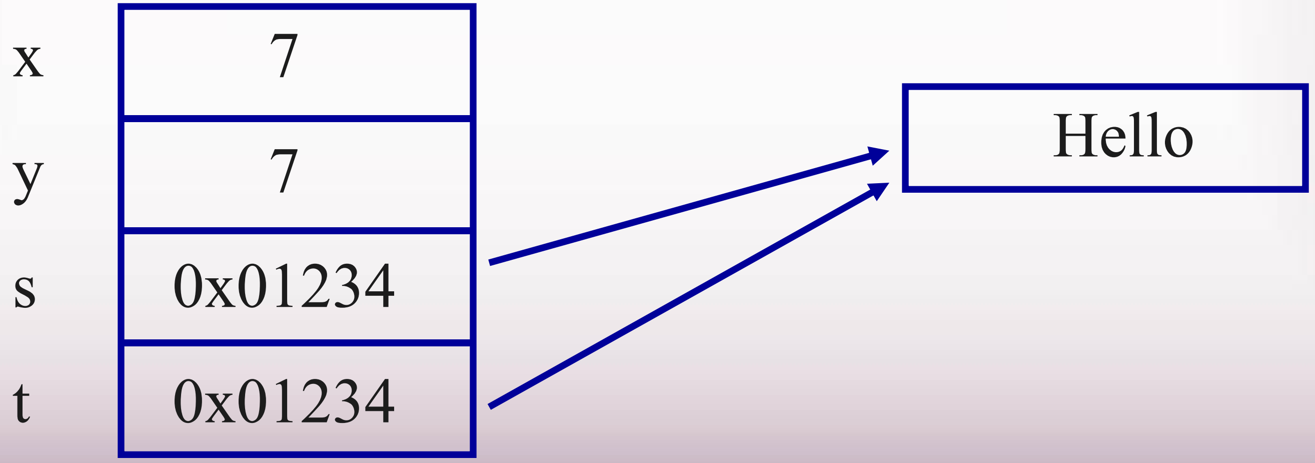
Age Should be Initialized before Use

# Instance Variable have Default Values

```
class Values
{
   private int a;
   private float b;
   private String c;
   public void display()
   {
       System.out.println("integer"+a);
       System.out.println("float"+b);
       System.out.println("String"+c);
}
public class DefaultVales {

   public static void main(String[] args) {

       Values v = new Values();
       v.display();
   }
}
```

# Assignment of reference variables

```
int x = 7;
int y = x;
String s = “Hello”;
String t = s;
```



x 7

y 7

s 0x01234

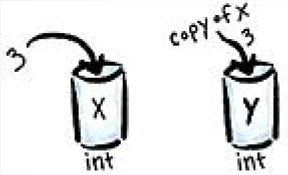
t 0x01234

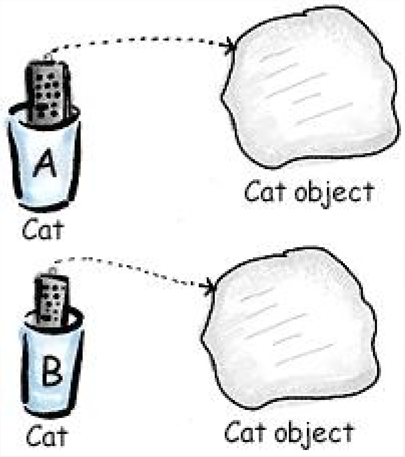
Hello

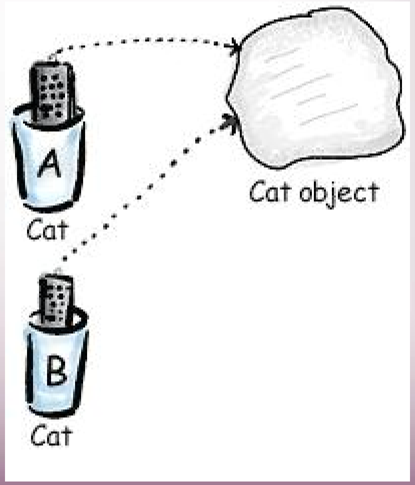
# Pass-by-Value

- The Java programming language only passes arguments by value for primitive data types.
- When an object instance is passed as an argument to a method, the value of the argument is a *reference* to the object
- The *contents* of the object can be changed in the called method, but the object reference is never changed
- **For primitives, you pass a copy of the actual value.**
- **For references to objects, you pass a copy of the reference**
- You never pass the object. All objects are stored on the heap.

# Pass By Value







# Casting Primitive Types

- Casting creates a new value and allows it to be treated as a different type than its source
- JVM can implicitly promote from a narrower type to a wider type
- To change to a narrow type explicit casting is required
- Byte -> short -> int -> long -> float -> double

# Casting Primitive Types

```
  public static void main (String [] args){
             int x = 99;
             double y = 5.77;
             x = (int)y;  //Casting
             System.out.println("x = "+ x);

               double y1 = x; //No Casting
    int i =42;
    byte bt;
    bt= (byte)i;

  System.out.println("The Short number"+ bt);
}
```

# Wrapper Classes

- Primitives  have no associated methods
- Wrapper Classes are used encapsulate primitives
- They also provide some static utility methods to work on them

| Primitive Type | Wrapper class |
|---|---|
| -boolean | Boolean |
| -byte | Byte |
| -char | Character |
| -double | Double |
| -float | Float |
| -int | Integer |
| -long | Long |
| short | Short |

# Wrapping Primitives

- Wrapping a value
  - int i = 288
  - Integer iwrap = new Integer(i);

- unWrapping a value
  - int unwrapped = iwrap.intValue();

- **Methods In Wrapper Class**
  - parseXxx()
  - xxxValue()
  - valueOf()

# Wrapper Class Method
# Convert String to Numbers

```
    public static void main(String args[])
         {
  int ino=Integer.parseInt(args[0]);
  float fno = Float.parseFloat(args[1]);
  double dno = Double.parseDouble(args[2]);
  Long lno = Long.parseLong(args[3]);

   System.out.println("Integer value" +ino );
  String strIno = Integer.toString(ino);
   System.out.Println(“String Value”+strIno);
   }
}
```

# Auto Boxing

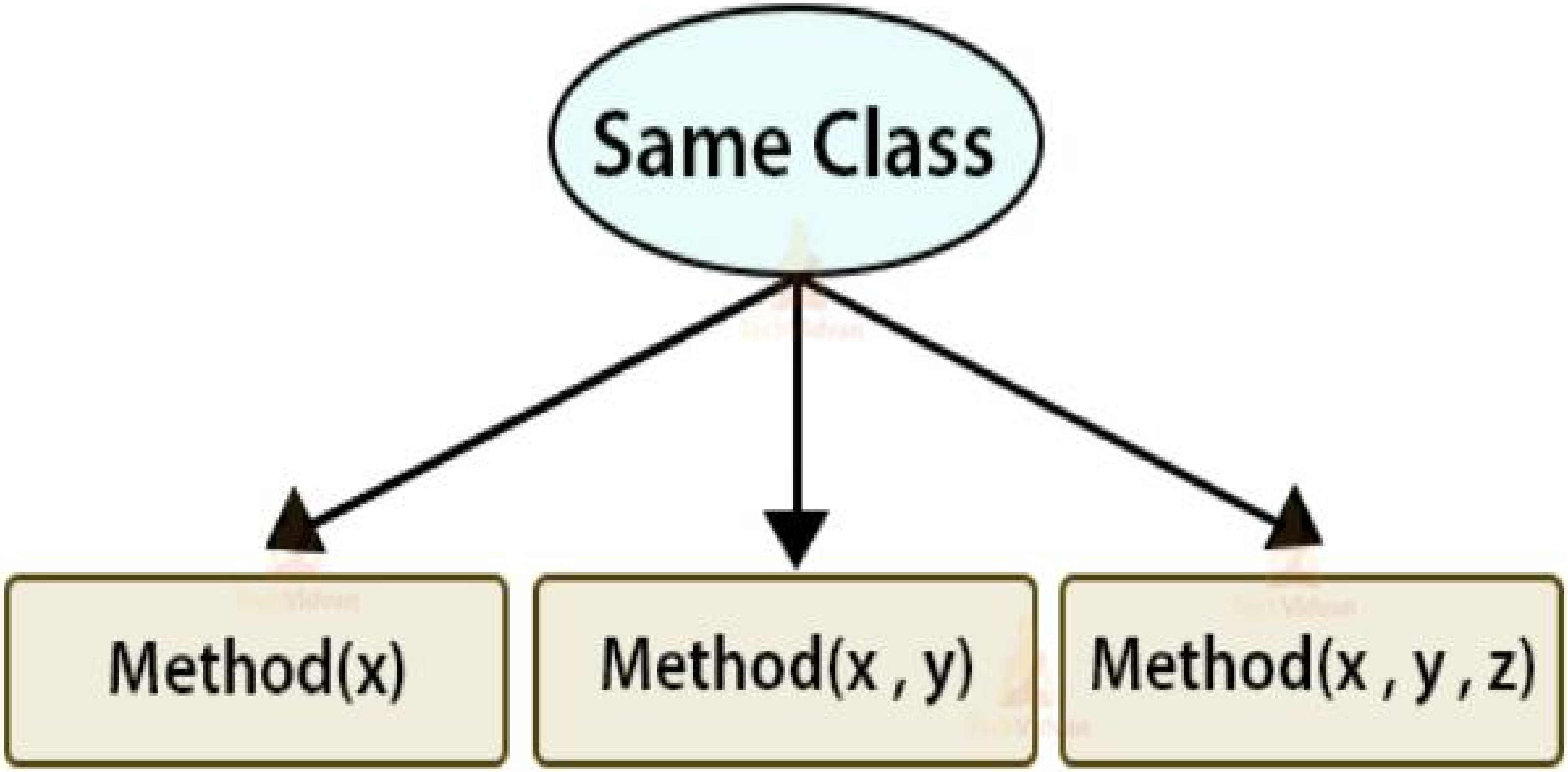
- Java 5.0 provided autoboxing

| | |
|---|---|
| Integer n = new Integer(123)<br>Int m = n.intValue()<br>m++;<br>n=new Integer(m);<br>System.out.println(n); | Integer n = new Integer(123);<br>n++;<br>System.out.println(n); |

# Auto Boxing

```
public class ABoxing {

     public void show(int a, float b)
     {
       System.out.println("Integer"+a*2);
       System.out.println("Float"+b*2);
     }

     public static void main(String[] args) {

       ABoxing abObj =new  ABoxing ();

       Integer a = 10;
       Float b =20f;
       abObj.show(a,b);

     }
   }
```

# Method Overloading in Java



# Method Overloading

- Methods of a class have the same name but different signatures
  - Similar behavior but for different data types.
- **Signature**
  - Name of the method and the number and types of formal parameters in particular order.
- They are independent methods of a class
  - Can call each other just like any other method.
- A method *can* be overloaded in the same class or in a subclass.

# Overloading Methods

- Overloaded methods MUST change the argument list.
- Overloaded methods CAN change the return type.
- Overloaded methods CAN change the access modifier.
- Overloaded methods CAN declare new or broader checked exceptions.

# Overloading and AutoBoxing

```
public class Overloading {



    public Integer add(Integer a , Integer b)
    {
            Integer c = a+b;

            return c+100;
    }

    public int add(int a , int b)
    {
      return a+b;
    }
```

# Application

```
public static void main(String[] args) {

        Overloading olObj = new Overloading();

        System.out.println(olObj.add(45, 55));



    }
```

**Output will be 100 and Not 200**

# Static Variables and Methods

- A static method belongs to a class and not to its instance objects and hence they are shared between Objects
- Static Methods can not be overridden
- They can only call other *static* methods
- They can access only static variables and not instance Variables
- They cannot refer to *this* or *super*
- **Instance variable : 1 per instance** and **Static variable : 1 per class**
- Static final variables are constants
- *main( )* is defined to be a static method so as to be invoked directly

# Static Method access only static

```
public class StatClass {

  private int id;
  private static String name;

  private void instMethod()
  {
    staticMethod();
    System.out.println(id);
    System.out.println(name);
  }
  private static void staticMethod()
  {
  System.out.println(id);
  System.out.println(name);
  instMethod();
  }
}
```

Can Access Static from Instance Method

Cannot Access Instance Variable From Static

Cannot Access Instance Method From Static

# Static Import

```
import java.util.*;
import static java.lang.System.out;
import static java.lang.System.in;

public class StaticImport {

public static void main(String[] args) {

Scanner kb = new Scanner(in);
out.print("Enter an integer ");

int x = kb.nextInt();
out.print("Enter a double ");

double d = kb.nextDouble();
out.println("The sum is " + (x+d));
}
}
```

# java.util.Scanner Class

- A simple text scanner which can parse primitive types and strings using regular expressions.
- A Scanner breaks its input into tokens using a delimiter pattern, which by default matches whitespace.
- The resulting tokens may then be converted into values of different types using the various next methods.

```
Scanner sc = new Scanner(System.in);
 int i = sc.nextInt();
  sc.close();
```

# java.util.Scanner Class

```
Scanner sc = new Scanner(System.in);

System.out.println("Enter The Number");
int number = sc.nextInt();
System.out.println("Enter the Name");
String name = sc.next();
System.out.println(number + ":"+name);
```

# java.util.Scanner Class

```java
public static void main(String[] args) {


           String line="Java,is,in,OOP,Language";

           Scanner sc1 = new Scanner(line);

           sc1.useDelimiter(",");

         while  (sc1.hasNext())
         {
            System.out.println(sc1.next());
         }


   }
```

# Flow Control

# Control Structures

- To change the ordering of the statements in a program
- Two types of Control Structures
- decision control structures ,
  - allows us to select specific sections of code to be executed
    - *if -- else , if – else if*
    - *switch -- case*
- repetition control structures
  - allows us to execute specific sections of the code a number of times
    - *while*
    - *do -- while*
    - *for*

# Decision Control Structures

- Types:
  - if-statement
  - if-else-statement
  - If-else if-statement

- If Specifies that a statement or block of code  will be executed if and only if a certain **boolean** statement is true.

```
       if( boolean_expression )
       statement;
or
       if( boolean_expression ){
       statement1;
       statement2;
       }
```

- **boolean_expression** : can be an expression or variable.

# if-else statement

```
if( boolean_exp ){
Statement(s)
}
else {
Statement(s)
}
```

```
if(boolean_exp1 )
statement1;
else if(boolean_exp2)
statement2;
else
statement3;
```

- For Comparison == used instead =
- = being an assignment operator
- equals Method Should Be Used for Objects comparison

# switch-statement

- Allows branching on multiple outcomes.
- switch expression  is an integer ,character expression or variable
- case_selector1, case_selector2 and unique integer or character constants.
- If none of the cases are satisfied, the optional default block if present is executed.

```
switch( switch_expression ){
case case_selector1:
Statement(s);
break;
case case_selector2:
Statement(s);
break;
default:
statement1;
}
```

# switch-statement

- When a switch is encountered,
  - evaluates the switch_expression,
  - jumps to the case whose selector matches the value of the expression.
  - executes the statements in order from that point on until a break statement is encountered
  - break statement stops executing statements in the subsequent cases, it will be  last statement in a case.
  - Used to make decisions based only on a single integer or character value.
  - The value provided to each case statement must be unique

# switch-statement

```
public double CalculateDiscount(int pCode)
{
double discountPercentage=0.0d;

switch(pCode)
{
case 1:
   discountPercentage=0.10d;
   break;
case 2:
   discountPercentage=0.15d;
   break;
case 3:
   discountPercentage=0.20d;
   break;
default:
   discountPercentage=0.0;
}
return discountPercentage;
}
```

# Switch-Case in a Method

- Can have Either Return or Break if present inside a Method, but should provide  a default Value

```
public String switchExample(int key)       {
        switch (key) {
        case 1:
                return "Good Morning";
        case 2:
                return "Good AfterNoon";
        case 3:
           return "Good Evening";
        default:
                return "Good Night";
        }
```

# Switch-Case Java 7.0

- From In Java SE 7 and later, can use a String object in the switch statement's expression.
- The String in the switch expression is compared with the expressions associated with each case label as if the String.equals method were being used.
- Will Throw a Null pointer Expression  if the expression is NULL

```
String color = "red";
switch (color) {
case "red":
System.out.println("Color is Red");
 break;
default:
System.out.println("Color not found");
}
```

# Repetition Control Structures

- **while-loop**
  - The statements inside the while loop are executed as long as the Boolean expression evaluates to true

- **do-while loop**
  - statements inside a do-while loop are executed several times as long as the condition is satisfied, the statements inside a do-while loop are executed at least once

```
while(boolean_expression){
statement1;
statement2;
}
```

Watch this

```
do{
statement1;
statement2;
}while(boolean_expression);
```