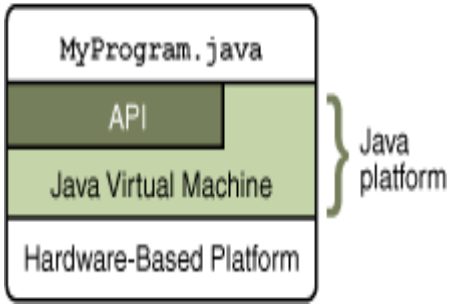
Core Java

Lesson Objective

- Introduction to Java
- Features of Java
- Evolution of Java
- Developing software in Java
- Writing, Compiling and execution of Simple Java Program
- Java Language Fundamentals

Introduction to Java

- **History of Java :**
  Developed: 1991 - Part of Sun's "Green Team Project
  Created by: Patrick Naughton, Mike Sheridan, and James Gosling
  Demo released: 1992
  Released: 1995
  Original name: Oak



- **Recent News:**

* 2006 – Sun released much of java as open source software
* 2009 – 2010 – Oracle corporation acquired Sun Microsystems

- **About Java Technology:**
  Java is a high level programming language created by Sun Microsystems.
  Java technology is **both a programming language** and a **platform**

Features of Java

- Object Oriented : class, object, inheritance, polymorphism … etc
- Simple : easy to learn
- Robust : strongly typed
- Secure : Byte code verifier, Security Manager …etc.
- Architecture Neutral: Platform independent
- Interpreted and Compiled
- Multithreaded: Concurrent running tasks
- Dynamic : run time linking of code [applet]
- Memory Management and Garbage Collection

Developing software in Java

- **Java Development Kit(JDK)**

  The Java Developer's Kit is distributed by Sun Microsystems. The JDK contains documentation, examples, installation instructions, class libraries packages and tools.

  There are two installers available.
  - JDK – Software development Kit – compile and run java program
  - JRE – Java Runtime Environment – run java program [subset of JDK]

- **How to Install Java?**
  - Download the latest Java Installable from

  http://www.oracle.com/technetwork/java/javase/downloads/index.html

- Java Virtual Machine (JVM) : Platform Independent

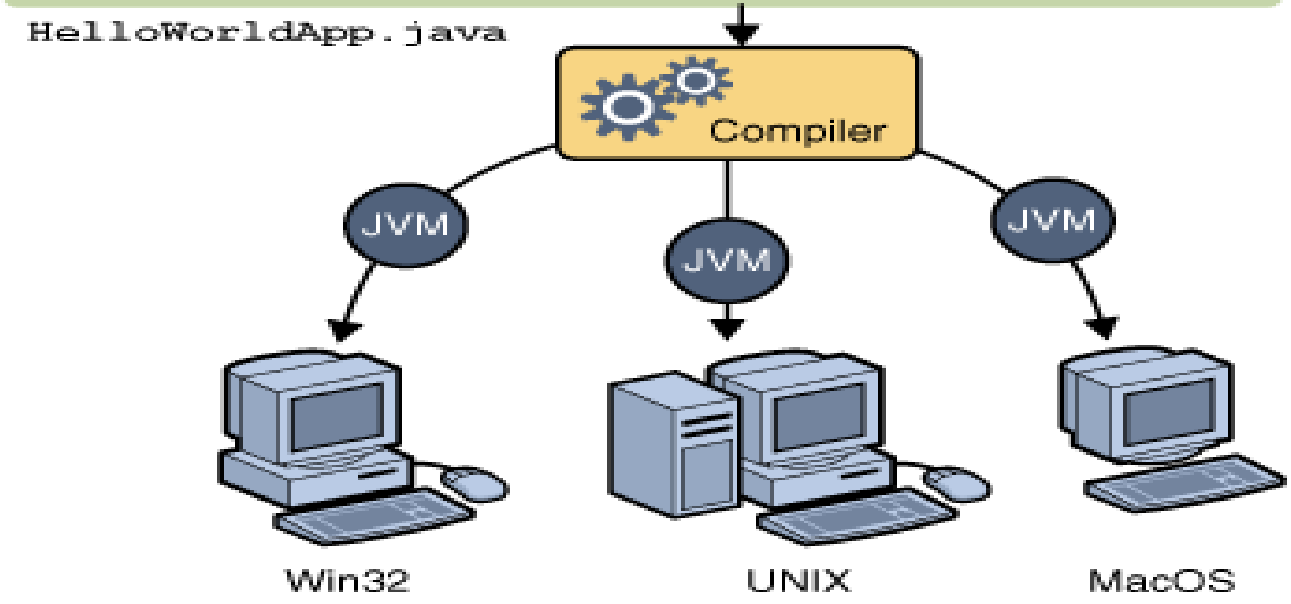
Writing, Compiling and Execution of Java Program

- Java Virtual Machine (JVM)
  - Java VM is available on many different operating systems. Hence the same .class files can be run on Microsoft Windows, the Solaris ™ Operating System (Solaris OS), Linux, or Mac OS.

Java Program

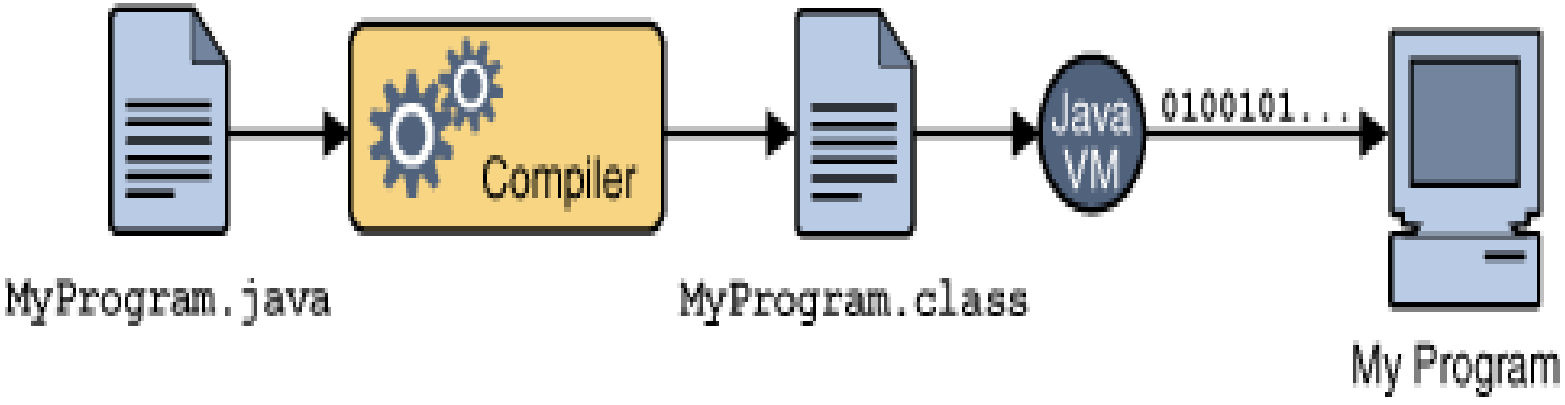
```
class HelloWorldApp {
      public static void main(String[] args) {
            System.out.println("Hello World!");
      }
}
```

HelloWorldApp.java



Writing, Compiling and Execution of Java Program

- A Java program is written, compiled and executed in three steps.
  - Step – 1: The source code is written in plain text file ending with .java extension.
  - Step – 2 : The source files are compiled into .class files using javac compiler.
  - Step – 3: The java launcher tool runs the application with an instance of JVM - Java Virtual Machine.



Writing, Compiling and Execution of Java Program

- **A Simple Java Program**

```
class ExampleProgram {
        public static void main(String[] args){
                System.out.println("I'm a Simple Program");
        }
}
```

- **Compiling the Program**

  javac ExampleProgram.java

- **Interpreting and Running the Program**

  Once the program successfully compiles into Java bytecodes, the Java interpreter is invoked at the command line on Unix and DOS shell operating systems as follows:

  java ExampleProgram

- Output : **I'm a Simple Program**

Java Language Fundamentals

- **Java Comments**
  - **Single line Comment** `// comment statement`
  - **Multiline Comment**
```
/* comment statement 1
   comment statement 2
*/
```

```
/* This Java class example describes how class is defined and being used in
 Java language.
*/
public class JavaClassExample {
                    //main method – Entry Point
    public static void main(String args[]){
          //method call
          System.out.println("Hello World");
    }
}
```

OUTPUT : Hello World

Java Comments

- **Comments :**

  Comments are descriptions that are added to a program to make code easier to understand. The compiler ignores comments and hence its only for documentation of the program. Java supports three comment styles.

- **Implementation Comments:**

Line style comments - begin with // and terminate at the end of the line.

Block style comments - begin with /* and terminate with */ that spans multiple lines.

- ***Documentation* comments**

  Begin with /** and terminate with */ that spans multiple lines. They are generally created using the automatic documentation generation tool, such as javadoc.

- Variables have a **data type**, that indicates the kind of value they can store.

Type	Size/Format	Description
byte	8-bit	Byte-length integer
short	16-bit	Short Integer
int	32-bit	Integer
long	64-bit	Long Integer
float	32-bit IEEE 754	Single precision floating point
double	64-bit IEE 754	Double precision floating point
char	16-bit	A single character
boolean	1-bit	True or False

Types of Variables

- Basic storage in a Java program
- Three types of variables:
  - Instance variables
    - Instantiated for every object of the class.
  - Static variables
    - Class Variables
    - Not instantiated for every object of the class.
  - Local variables
    - Declared in methods and blocks.
- Formal Parameters: Arguments passed to a function.

Types of Variables

```
class emp {
   String name;   //Instance Variable
   int empcode;  //Instance Variable
   float basicsalary;        //Instance Variable
   static int obcount;    //static variable
   void calculateSalary() {
   float temp sal;   //Local variable
   ....
   }
    } //class emp ends
```

Methods and Parameter Passing

- Parameters or arguments passed to a function are passed by value for primitive data-types.

- Parameters or arguments passed to a function are passed by reference for non-primitive data-types
  - Example: All Java objects.

```
void myFunction(Integer a) {
            a++;
            System.out.println(a);
}
....
Integer x = 45;
System.out.println(x);
myFunction(x);
System.out.println(x);

Output is, “45 46 45”
```

```
void myFunction(int a) {
            a++;
            System.out.println(a);
}
....
int x = 45;
System.out.println(x);
myFunction(x);
System.out.println(x);

Output is, “45 46 45”
```

Variable Argument List

- New feature added in J2SE5.0.
- Allows methods to receive unspecified number of arguments.
- An argument type followed by ellipsis(…) indicates variable number of arguments of a particular type.
  - *Variable-length* argument can take from *zero* to *n* arguments.
  - Ellipsis can be used only once in the parameter list.
  - Ellipsis must be placed at the end of the parameter list.

- Valid Code

```
void print(int a,int y,String...s)
  {
        //code
  }
```

print(1,1,"XYZ") or print(2,5) or print(5,6,"A","B") invokes the above print function

- Invalid Code

```
void print(int a, int b...,float c)
{
   //code
}
```

Demo:

- varargs.java

varargs.java

Keywords in Java

- Keywords are reserved words can't be used as variables, function and class names.

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

* not used
** added in 1.2
*** added in 1.4
**** added in 5.0

Operators and Assignments Java

- Operators can be divided into following groups:
  - Arithmetic
  - Bitwise
  - Relational
  - Logical
  - *instanceOf* Operator

Arithmetic Operators

Operator	Result
+	Addition
-	Subtraction (or unary) operator
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
-=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
--	Decrement

Bitwise Operators

- Apply upon *int*, *long*, *short*, *char* and *byte* data types:

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment

Relational Operators

- Determine the relationship that one operand has to another.
  - Ordering and equality.

Operator	Result
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Logical Operators

Operator	Result
&&	Logical AND
||	Logical OR
^	Logical XOR
!	Logical NOT
==	Equal to
?:	Ternary if-then-else

instanceOf Operators

- The *instanceof* operator compares an object to a specified type
- Checks whether an object is:
  - An instance of a class.
  - An instance of a subclass.
  - An instance of a class that implements a particular interface.
  - The following returns *true*:

```
new String("Hello") instanceof String;
```

Control Statements

- Use control flow statements to:
  - Conditionally execute statements.
  - Repeatedly execute a block of statements.
  - Change the normal, sequential flow of control.

- Categorized into two types:
  - Selection Statements
  - Iteration Statements

Selection Statements

Allows programs to choose between alternate actions on execution.

“If” used for conditional branch:

if (condition) statement1;

- **else statement2;**

“Switch” used as an alternative to multiple “if’s”:

switch(expression){

```
case value1:      //statement sequence
      break;
case value2:
                  //statement sequence
      break; …
default:
                  //default statement sequence
}
```

Iteration Statements

- Allow a block of statements to execute repeatedly.
- *While* Loop:
  - Enters the loop if the condition is *true*.

```
while (condition)
{  //body of loop }
```

- *Do – While* Loop:
  - Loop executes at least once even if the condition is *false*.

```
do
{ //body of the loop
} while(condition)
```

- *For* Loop:

```
for( initialization ; condition ; iteration)
{ //body of the loop }
```

Enhanced for Loop (foreach)

- New feature introduced in Java 5.

- Iterate through a collection or array.
  - Syntax:

```
for (variable : collection)
{ //code}
```

  - Example

```
int sum(int[] a)
{
 int result = 0;
for (int i : a)
result += i;
return result; }
```

Objects and Classes

- Classes:
  - A template for multiple objects with similar features.
  - A blueprint or the definition of objects.

```
class < class_name>
{
   type var1; …
   Type method_name(arguments )
   {
            body
   } …
} //class ends
```

- Objects:
  - Instance of a class.
  - Concrete representation of class.

Introduction to Classes

- Code Snippet:

```
class MyBox{

        double width;
        double height;
        double depth;

        double volume()
        {
                return width*height*depth;
        } //method volume ends.

}//class Mybox ends.
```

Declaring Objects

- Code Snippet

```
class Impl{
        public static void main(String a[])
        {
                //declare a reference to object
        MyBox b;

                //allocate a memory for box object.
        b = new MyBox();

                // call a method on that object.
        b.volume();
       }
}
```

Packages in Java

- A Java package is a set of classes which are grouped together.
  - To organize Java classes
  - To code multiple Java classes with the same name.
- **Using Java Packages :**

  To use a package in your Java source code, you must either import the package or use the fully qualified class name each time you use a class.

- **Importing Classes :**

```
import java.util.Date;
                Or
Java.util.Date d = new java.util.Date();
```

All the classes in Java.lang package are automatically imported when the program runs. The rest of other classes in the packages are to be imported by the programmer by using the import keyword.

Demo:

Box.java

Constructors

- Constructors are similar to methods except that constructors have the *same name as the class* . When a new instance (a new object) of a class is created using the new keyword, the *constructor* for that class is called. Constructors are used to initialize the instance variables (fields) of an object.

- **Default constructor**. If we don't define a constructor for a class, a *default parameterless constructor* is automatically created by the compiler. The default constructor calls the default parent constructor (super()) and initializes all instance variables to default value (zero for numeric types, null for object references, and false for booleans).

- **Default constructor is created only if there are no constructors**. If you define *any* constructor for your class, no default constructor is automatically created.

- **Differences between methods and constructors**.
  - Constructor does not have a return type
  - There is *no return statement* in the body of the constructor.
  - The *first line* of a constructor must either be a call on another constructor in the same class (using this), or a call on the superclass constructor (using super). If the first line is neither of these, the compiler automatically inserts a call to the parameterless super class constructor.

Constructor – Example program

Example of explicit this constructor call

```
public class Point {
        int m_x;
         int m_y;
        //============ Constructor public
        Point(int x, int y) {
                m_x = x;
                m_y = y;
        }
        //============ Parameterless default constructor
        public Point() {
                this(0, 0);
        // Calls other constructor.
        }
. . . }
```

Types of Class Members

- Default access members (No access specifier)
  - accessible only by classes in the same package
- Private members
  - cannot be accesses by anywhere outside the enclosing class
- Public members
  - are visible to any class in the Java program, whether these classes are in the same package or in another package.
- Protected member
  - only accessible by subclasses in same aa well as other packages

Memory Management

- Dynamic and Automatic
- No *Delete* operator
- Java Virtual Machine (JVM) de-allocates memory allocated to unreferenced objects during the garbage collection process.

Enhancement in Garbage Collector

- Garbage Collector:
  - Lowest Priority Daemon Thread
  - Runs in the background when JVM starts.
  - Collects all the unreferenced objects.
  - Frees the space occupied by these objects.
  - Call *System.gc()* method to "hint" the JVM to invoke the garbage collector.
    - There is no guarantee that it would be invoked. It is implementation dependent.

Memory Management

- All Java classes have *constructors*.
  - Constructors initialize a new object of that type.

- Default no-argument constructor is provided if program has no constructors.
  - Constructors:
    - Same name as the class.
    - No return type; not even void.

Finalize() Method

- Memory is automatically de-allocated in Java.
- Invoke *finalize()* to perform some housekeeping tasks before an object is garbage collected.
- Invoked just before the garbage collector runs:
  - protected void finalize()

- A group of like-typed variables referred by a common name.

- Array declaration:
  - int arr [];
    arr = new int[10]
  - int arr[] = {2,3,4,5};
  - int two_d[][] = new int[4][5];

Creating Array Objects

- Arrays of objects too can be created:
  - Example 1:
    - Box Barr[] = new Box[3];
    - Barr[0] = new Box();
    - Barr[1] = new Box();
    - Barr[2] = new Box();
  - Example 2:
    - String[] Words = new String[2];
    - Words[0]=new String(“Bombay”);
    - Words[1]=new String(“Pune”);

Creating Array Objects

- Arrays of objects too can be created:
  - Example 1:
    - Box Barr[] = new Box[3];
    - Barr[0] = new Box();
    - Barr[1] = new Box();
    - Barr[2] = new Box();
  - Example 2:
    - String[] Words = new String[2];
    - Words[0]=new String(“Bombay”);
    - Words[1]=new String(“Pune”);

Demo：Creating Array Objects

Demo: ArrayDemo.java

Static variable

- Static variable is shared by all the class members.
- Used independently of objects of that class.
- Static members can be accessed before an object of a class is created, by using the class name:
  - Example:

```
static int intNum1 = 3;
```

Static Methods

- Restrictions:
  - Can only call other static methods.
  - Must only access other static data.
  - Cannot refer to *this* or *super* in any way.
  - Can not access non-static variables and methods directly:
    - Explicit instance variables should be made available to the method.
    - Method *main()* is a static method. It is called by JVM.

The Object Class

- Cosmic super class.
- Ultimate ancestor
  - Every class in Java implicitly extends Object.
- Object type variables can refer to objects of any type:
  - Example:

```
Object obj = new Emp();
```

- Object Class Methods:
  - void finalize()
  - Class getClass()
  - String toString()

Object Class Methods

Method	Description
`boolean equals(Object)`	Determines whether one object is equal to another
`void finalize()`	Called before an unused object is recycled.
`class getClass()`	Obtains the class of an object at run time.
`int hashCode()`	Return the hashcode associated with the invoking object.
`String toString()`	Returns a string that describes the object

The System Class

- Used to interact with any of the system resources.
- Cannot be instantiated.
- Contains a methods and variables to handle system I/O.
- Facilities provided by the System class:
  - Standard input
  - Standard output
  - Error output streams

The System Class (contd..)

Method	Description
`void currentTimeMillis()`	Returns the current time in terms of milliseconds since midnight, January 1, 1970
`void gc()`	Initiates the garbage collector.
`void exit(int code)`	Halts the execution and returns the value of integer to parent process usually to an operating system.

Demo : The System Class

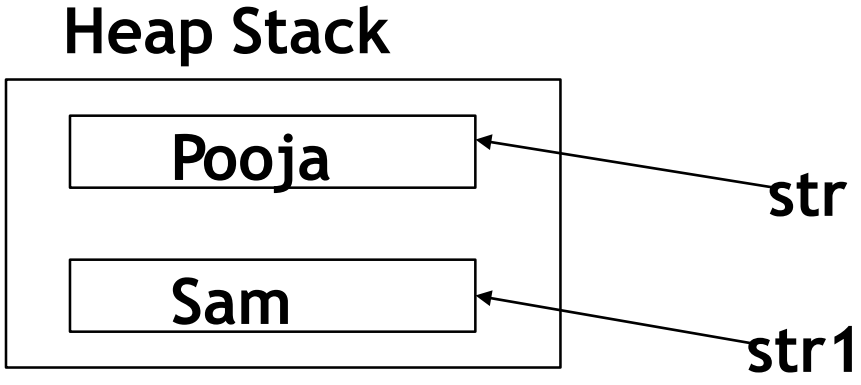
Demo: Elapsed.java

String Handling

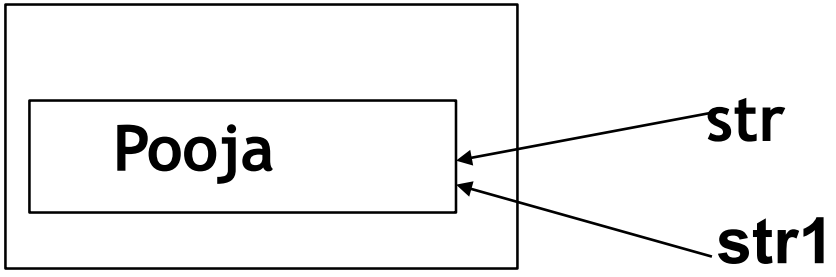
- String is handled as an object of class String and not as an array of characters.
- String class is a better and a convenient way to handle any operation.
- One main restriction is that once an object of this class is created, the contents cannot be changed.

- length(): length of string.
- indexOf(): searches an occurrence of a char, or string within other string.
- substring(): Retrieves substring from the object.
- trim(): Removes spaces.
- valueOf(): Converts data to string.

String str = new String(“Pooja”);
String str1 = new String(“Sam”);



String str = new String(“Pooja”);
String str1 = str;



String Concatenation

- Use a “+” sign to concatenate two strings:
  - String Subject = "Core " + "Java"; -> Core Java
  - *String concatenation* operator if one operand is a string:
    - String a = "String"; int b = 3; int c=7
    - System.out.println(a + b + c); -> String37
  - *Addition* operator if both operands are numbers:
    - System.out.println(a + (b + c)); -> String10

String Concatenation (contd..)

- public String concat(String s)
  - Used to concatenate a string to an existing string.
  - String x = "Core ";
  - System.out.println(x=x.concat(" Java"));
  - Output -> "Core Java”

String Comparison

```
class EqualsNotEqualTo {
   public static void main(String args[]) {
         String str1 = "Hello";
         String str2 = new String(str1);
System.out.println(str1 + " equals " + str2 + " -> " +
   str1.equals(str2));

 System.out.println(str1 + " == " + str2 + " -> " + (str1 ==str2));
   }
}
Output :  Hello equals Hello -> true
              Hello == Hello -> false
```

StringBuffer Class

- Use the following to make ample modifications to character strings:
  - *java.lang.StringBuffer*
  - java.lang.StringBuilder

- Many string object manipulations end up with a many abandoned string objects in the *String pool*
  - String Objects are immutable.

```
StringBuffer sb = new StringBuffer("abc");
sb.append("def");
System.out.println("sb = " + sb); // output is "sb = abcdef"
```

StringBuilder Class

- Added in Java 5.

- Exactly the same API as the *StringBuffer* class, except:
  - It is not thread safe.
  - It runs faster than StringBuffer.

```
StringBuilder sb = new StringBuilder("abc");
sb.append("def").reverse().insert(3, "---");
System.out.println( sb ); // output is "fed---cba"
```

Wrapper Classes

- Correspond to primitive data types in Java.
- Represent primitive values as objects.
- Wrapper objects are immutable.

Simple Data Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
char	Character
float	Float
double	Double
boolean	Boolean
void	Void

Casting for Conversion of Data type

- Casting operator converts one variable value to another where two variables correspond to two different data types.

```
variable1 = (variable1) variable2
```

- Here, *variable2* is typecast to *variable1*.

- Data type can either be a *reference* type or a *primitive* one.

Casting Between Primitive Types

- When one type of data is assigned to another type of variable, *automatic type conversion* takes place if:
  - Both types are compatible.
  - Destination type is larger than the source type.
  - No explicit casting is needed (widening conversion).

```
int a=5; float b; b=a;
```

- If there is a possibility of data loss, explicit cast is needed:

```
int i = (int) (5.6/2/7);
```

Casting Between Reference Types

- One class types involved must be the same class or a subclass of the other class type.
- Assignment to different class types is allowed only if a value of the class type is assigned to a variable of its superclass type.
- Assignment to a variable of the subclass type needs explicit casting:

```
String StrObj = Obj;
```

- Explicit casting is not needed for the following:

```
String StrObj = new String("Hello");
Object  Obj = StrObj;
```

Casting Between Reference Types (contd..)

- Two types of reference variable castings:
  - Downcasting:

```
Object Obj = new Object ( );
String StrObj = (String) Obj;
```

  - Upcasting:

```
String StrObj = new String("Hello");
Object  Obj = StrObj;
```