



Welcome to
Java Basics
Session

JavaTM



+91-9538998962

sandeep@knowbigdata.com

WELCOME - KNOWBIGDATA

- Interact - Ask Questions
- Real Life Project
- Lifetime access of content
- Quizzes & Certification Test
- Class Recording
- 10 x (3hr class)
- Cluster Access
- Socio-Pro Visibility
- 24x7 support
- Mock Interviews

ABOUT ME

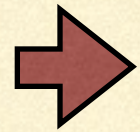
2014	KnowBigData	Founded
2014	Amazon	Built High Throughput Systems for Amazon.com site using in-house NoSql.
2012		
2012	InMobi	Built Recommender after churning 200 TB
2011	tBits Global	Founded tBits Global Built an enterprise grade Document Management System
2006	D.E.Shaw	Built the big data systems before the term was coined
2002	IIT Roorkee	Finished B.Tech somehow.
2002		



TODAY'S CLASS

- What is Java?
- Why Java?
- What is Hadoop?
- Components Hadoop
- HDFS Architecture
- NameNode + datanode
- Further Reading/Assignment

COURSE CONTENT



I

Session 1

II

Session 2

III

Session 3

JAVA?

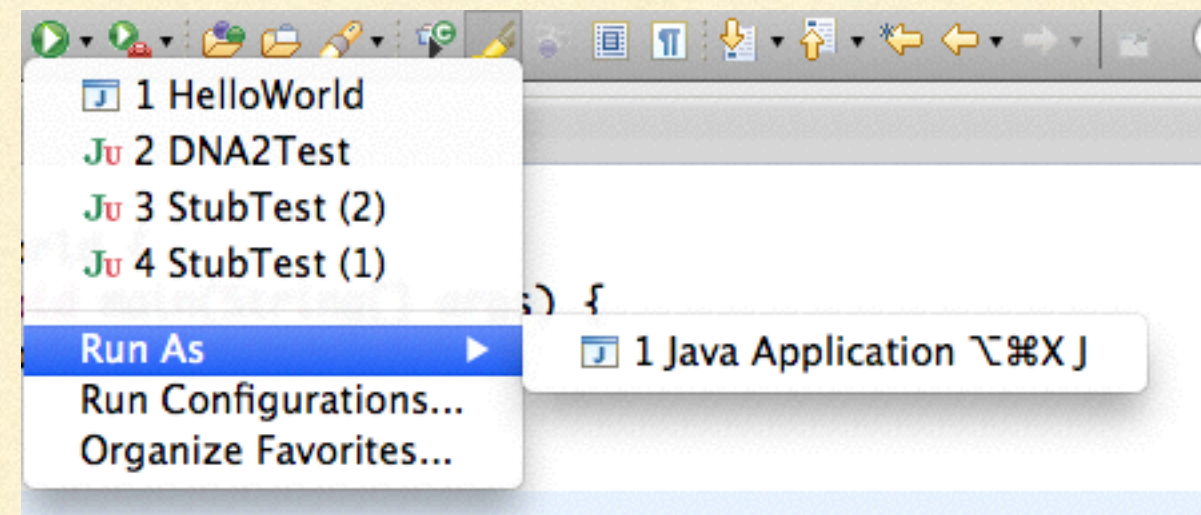
- High Level Programming language
- By Sun Microsystems in 1995
- Runs on Windows, Linux, Unix, Mac - intel/amd/...
- Object Oriented
- Portable
- Compiled - Robust
- Multithreaded / High Performance / Distributed
- Interpreted

Java Source == Compiler ==> Byte Code == JVM ==> Executes

GETTING STARTED - ECLIPSE WAY

- Download Eclipse for Java Developers
 - <https://eclipse.org/downloads/>
- Create a new project, File => Project => Java Project
- Specify any project name e.g. javatutorials
- Click Finish
- Right click on "Src" in recently created project in project explorer
- New -> Class and then specify a class name e.g. "HelloWorld"
- Create a main method and get going!!
- Run!

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



GETTING STARTED - COMMAND LINE WAY

- Install JDK: www.oracle.com/technetwork/java/javase/downloads/
- [opt] Get text editor like **Notepad++** / **Sublime** / **Emacs**
- Create HelloWorld.java with contents as previous
- Goto cmd prompt
- *cd* to directory having HelloWorld.java
- Compile
 - *javac HelloWorld.java*
 - This would create HelloWorld.class if successful
- Run
 - *java HelloWorld*
 - "Hello,All"

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

```
sgiri-air:src sravani$ javac HelloWorld.java  
sgiri-air:src sravani$ java HelloWorld  
Hello, All  
sgiri-air:src sravani$
```

Basic Syntax

- Case Sensitive
- Every file is a class
 - The name of the class should be same as the file name.
- Every class can have methods
 - e.g `public void myMethodName()`
- The method run by "java" program by default:
 - `public static void main(String args[])`

Identifiers

Class name, Method Name & Variable name

- Composed of
 - a letter ([A-Za-z])
 - dollar (\$)
 - an underscore (_).
 - Number
- Must Not Begin with
 - A number
- Can't be a **keyword**
- Are case sensitive

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



age, age2, \$salary, _value, __l_value



l23abc, -salary, #salary, ~salary, !age



Identifiers

Which of the following are invalid identifier?

- A. 2tuple
- B. _____X
- C. _____
- D. \$\$\$
- E. \$2
- F. SANDEEP
- G. sanDEEP
- H. .san
- I. san.giri
- J. san_giri

Identifiers

Which of the following are invalid identifier?

- ✗ A. 2tuple
- B. _____X
- C. _____
- D. \$\$\$
- E. \$2
- F. SANDEEP
- G. sanDEEP
- ✗ H. .san
- ✗ I. san.giri
- J. san_giri

```
public class HelloWorld {  
    public static void main(String[] args) {  
        int __ = 12;  
        int $x = 10;  
        int $$ = 10;  
        int $2 = 22;  
        System.out.println(__ + $x + $$ + $2);  
    }  
}
```

Ans: A, H, I

Modifiers

- Access Modifiers:
 - default
 - public
 - protected
 - private
- Non-access Modifiers:
 - final
 - abstract
 - strictfp

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

Keywords

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

Comments in Java

```
public class MyFirstJavaProgram{  
  
    /* This is my first java program.  
    * This will print 'Hello World' as the output  
    * This is an example of multi-line comments.  
    */  
  
    public static void main(String []args){  
        // This is an example of single line comment  
        /* This is also an example of single line comment. */  
        System.out.println("Hello World");  
    }  
}
```

A line containing only whitespace, possibly with a comment, is known as a blank line, and Java totally ignores it.

Primitive Types

- byte
 - 8 bit signed
 - -128 to 127
 - e.g. byte a = 10;
- short
 - 16 bit signed
 - (-2^{15}) to $(2^{15}-1)$
 - -32,768 to 32,767
 - e.g. short s = 10000,
 - e.g. short r = -20000
- int
 - 32 bit signed integer
 - -2^{31} to $2^{31}-1$
 - Ex. int a = 100000,
 - int b = -200000
- long
 - 64-bit signed integer
 - Ex. long a = 100L, long b = -20L
- float
 - 32-bit decimal number
 - float f1 = 234.5f
- double
 - 64-bit floating point
 - double d1 = 123.4
- boolean
 - true / false
 - default value is false
- char
 - single 16-bit Unicode character
 - char letterA = 'A'

Java Literals - source code representation of fixed value.

They are represented directly in the code without any computation.

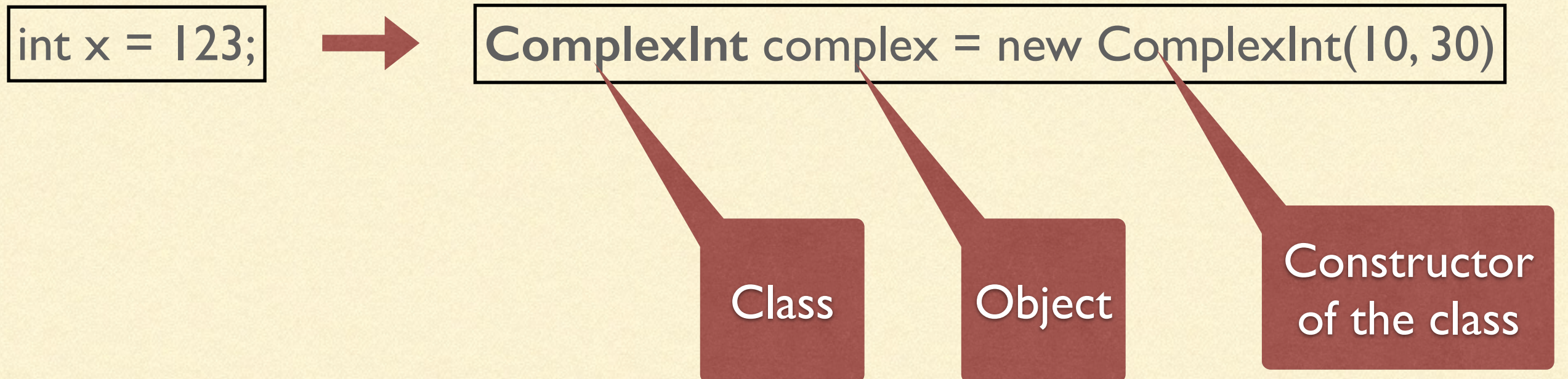
```
char a = 'A'  
int decimal = 100;  
int octal = 0144;  
int hexa = 0x64;
```

```
"Hello World"  
"two\nlines"  
"\\"This is in quotes\\""
```

Notation	Character represented
\n	Newline (0x0a)
\r	Carriage return (0x0d)
\f	Formfeed (0x0c)
\b	Backspace (0x08)
\s	Space (0x20)
\t	tab
\"	Double quote
\'	Single quote
\\	backslash
\ddd	Octal character (ddd)
\uxxxx	Hexadecimal UNICODE character (xxxx)

Classes & Objects

When primitives did not suffice, we create classes.

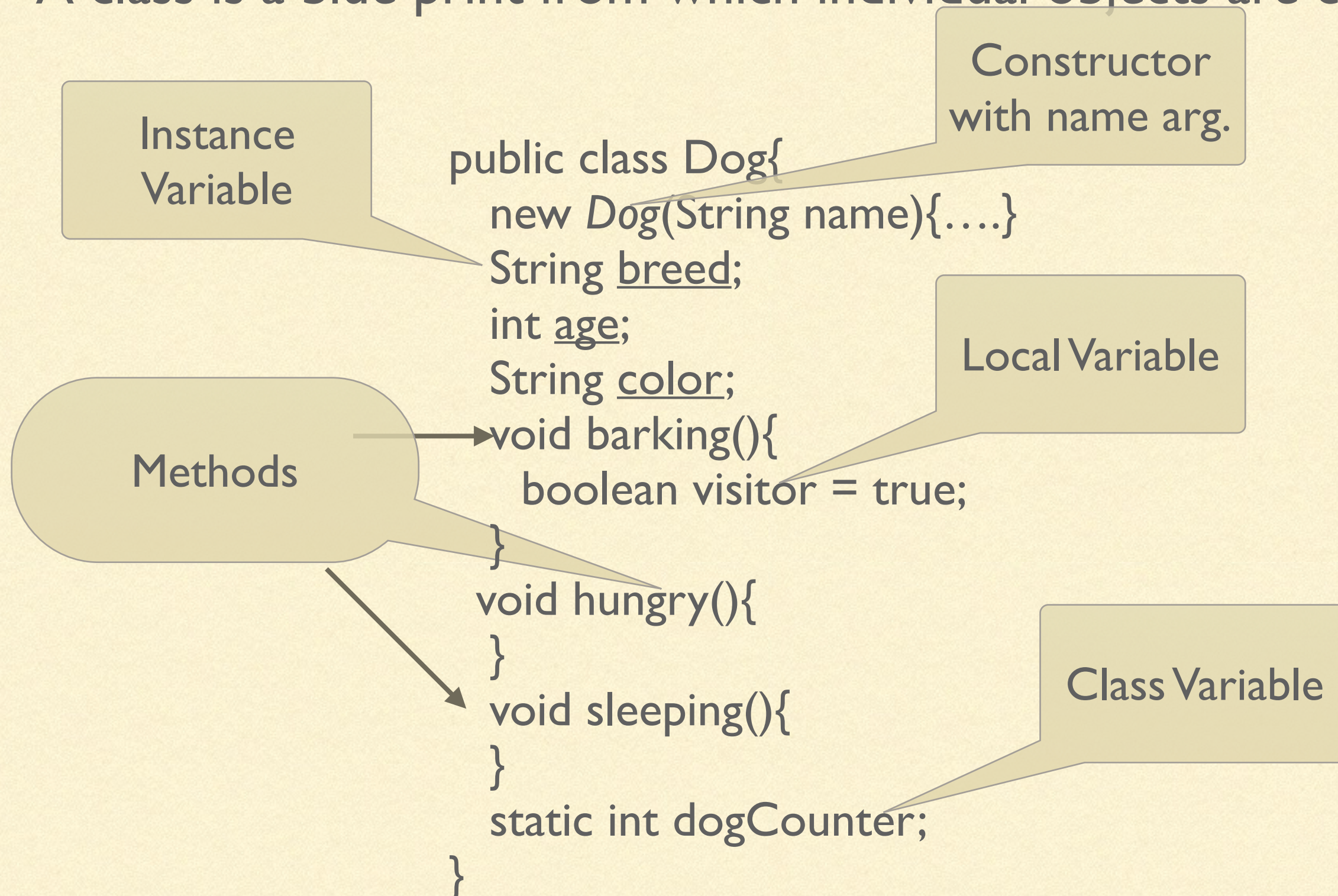


Objects have states and behaviors.

A class can be defined as a template/blue print that describes the behaviors/states that object of its type support.

Classes in Java

A class is a blue print from which individual objects are created.



Creating Objects

Object is created from a class using new keyword.

```
public class Puppy{  
    public Puppy(String name){  
        // This constructor has one parameter, name.  
        System.out.println("Passed Name is :" + name );  
    }  
}
```

```
public class PuppyX{  
    public static void main(String []args){  
        // Following statement would create an object myPuppy  
        Puppy myPuppy = new Puppy( "tommy" );  
    }  
}
```

Output: Passed Name is :tommy

Step 1: Declaration

A variable declaration with a variable name with an object type.



Step 2: Instantiation

The 'new' key word is used to create the object.



Step 3: Initialisation

The 'new' keyword is followed by a call to a constructor. This call initialises the new object.

Accessing Instance Variables and Methods

objectName.functionName(arguments)

```
public class Puppy{  
  
    int puppyAge;  
  
    public Puppy(String name){  
        // This constructor has one parameter, name.  
        System.out.println("Passed Name is :" + name );  
    }  
  
    public void setAge( int age ){  
        puppyAge = age;  
    }  
  
    public int getAge( ){  
        System.out.println("Puppy's age is :" + puppyAge );  
        return puppyAge;  
    }  
}
```

```
public class PuppyCaller{  
    public static void main(String []args){  
        /* Object creation */  
        Puppy myPuppy = new Puppy( "tommy" );  
  
        /* Call class method to set puppy's age */  
        myPuppy.setAge( 2 );  
  
        /* Call another class method to get puppy's age */  
        myPuppy.getAge( );  
  
        /* You can access instance variable as follows as well */  
        System.out.println("Variable Value :"  
                            + myPuppy.puppyAge );  
    }  
}
```

Variable Value: 2

Source file declaration rules

1. A source file can have multiple non public classes.
2. There can be only one public class per source file.
3. File name should be same as public class name

```
$cat Employee.java
public class Employee{
    ...
}
class Someother
{
    ....
}
```

Java Package

- Way of organising the code(classes, interfaces)
- First line in the class has definition
 - *package my.nice.package*
- Folder Structure should be same as the package name.
- A class X in package a.b.c
 - would be residing in a folder a/b/c
 - The name of the file would be X.java
- Declaring a package is optional but recommended
- Especially If two classes/interfaces with same name

Import Statements

- To use other classes you need to import them
- If they do not reside in current package
- You can import all classes in a package by ‘*’
 - Example: *import java.io.*;*
- Import statement come after the package declaration

Variable Types

- Named Storage in the RAM
- Can change the value
- Has a type - decides size and shape

`int a, b, c; // Declares three ints, a, b, and c.`

`int a = 10, b = 10; // Example of initialization`

`byte B = 22; // initializes a byte type variable B.`

`double pi = 3.14159; // declares and assigns a value of PI.`

`char a = 'a'; // the char variable a is initialized with value 'a'`

Modifiers

Access Control Modifiers - Who can access a variable.

- default - Visible to the package
- private - Visible to the class
- public - Visible to the world.
- protected - Visible to the package and all subclasses

Other Modifiers

- static - Controls creating class methods and variables
- final - to finalize implementations of classes, methods, variables.
- abstract - Creating abstract classes and methods.
- synchronized and volatile - for threads.

Basic Operators - The Arithmetic Operators

What is the output?

```
public class HelloWorld {  
    public static void main(String[] args) {  
        int A = 10, B = 20;  
        System.out.println( (A+B) / (A-B));  
        System.out.println((A++) % 5);  
        System.out.println(--B*1.0 / 5);  
    }  
}
```

Basic Operators - The Arithmetic Operators

What is the output?

```
public class HelloWorld {  
    public static void main(String[] args) {  
        int A = 10, B = 20;  
        System.out.println( (A+B) / (A-B));  
        System.out.println((A++) % 5);  
        System.out.println(--B*1.0 / 5);  
    }  
}
```

-3
0
3.8

Basic Operators - The Relational Operators

What is the output?

```
public class HelloWorld {  
    public static void main(String[] args) {  
        int A = 10, B = 20;  
        System.out.println(A == B);  
        System.out.println(A != B);  
        System.out.println(A > B);  
        System.out.println(A < B);  
        System.out.println(A >= (B-10));  
    }  
}
```


Basic Operators - The Relational Operators

What is the output?

```
public class HelloWorld {  
    public static void main(String[] args) {  
        int A = 10, B = 20;  
        System.out.println(A == B);  
        System.out.println(A != B);  
        System.out.println(A > B);  
        System.out.println(A < B);  
        System.out.println(A >= (B-10));  
    }  
}
```

false
true
false
true
true

Basic Operators - The Bitwise Operators

a = 60 = 0011 1100

b = 13 = 0000 1101

[AND - If both are one] a&b = 0000 1100

[OR - Either is one] a|b = 0011 1101

[XOR - Both are different] a^b = 0011 0001

[NOT - Opposite] ~a = 1100 0011

[Shifting the bits to the left, 120] a<<1 = 0111 1000

[Shifting the bits to the right, 30] a>>1 = 0001 1110

Basic Operators - The Logical Operators

&& - Logical And - True if both conditions are true.

$(4/2==2) \ \&\& \ (3+2 == 6)$

True && False

False

Basic Operators - The Logical Operators

|| - Logical Or - True if either conditions is true.

$(4/2==2) \quad || \quad (3+2 == 6)$

True || False

True

The Assignment Operators

```
int A = 10, B = 30;  
    A += 20;  
System.out.println(A); // ??  
    A /= 10;  
System.out.println(A); // ??  
    A *= 12;  
System.out.println(A); // ??  
    A %= 10;  
System.out.println(A); // ??
```

The Assignment Operators

```
int A = 10, B = 30;  
    A += 20;  
System.out.println(A); // 30  
    A /= 10;  
System.out.println(A); // 3  
    A *= 12;  
System.out.println(A); // 36  
    A %= 10;  
System.out.println(A); // 6
```


Conditional Operator (? :):

Single line if, else statement

Condition

Return this value
if condition is
true.

```
public class HelloWorld {  
  
    public static void main(String args[]) {  
        int a , b;  
        a = 10;  
        b = (a == 1) ? 20 : 30;  
        System.out.println( "Value of b is : " + b );  
  
        b = (a == 10) ? 20 : 30;  
        System.out.println( "Value of b is : " + b );  
    }  
}
```

Return this value
if condition is
false

Conditional Operator (? :):

Single line if, else statement

Condition

Return this value
if condition is
true.

```
public class HelloWorld {  
    public static void main(String args[]) {  
        int a , b;  
        a = 10;  
        b = (a == 1) ? 20: 30;  
        System.out.println( "Value of b is : " + b );  
  
        b = (a == 10) ? 20: 30;  
        System.out.println( "Value of b is : " + b );  
    }  
}
```

Return this value
if condition is
false

Value of b is : 30
Value of b is : 20

Precedence of Java Operators

Category	Operator	Associativity	Precedence
Postfix	() [] . (dot)	Left to right	Highest
Unary	++ -- ! ~	Right to left	
Multiplicative	* / %	Left to right	
Additive	+ -	Left to right	
Shift	>> >>> <<	Left to right	
Relational	> >= < <=	Left to right	
Equality	== !=	Left to right	
Bitwise AND	&	Left to right	
Bitwise XOR	^	Left to right	
Bitwise OR		Left to right	
Logical AND	&&	Left to right	
Logical OR		Left to right	
Conditional	?:	Right to left	
Assignment	= += -= *= /=	Right to left	
Comma	,	Left to right	Lowest

$2 * 3 / 2$

$3 / 2 * 2$

$2 + 3 / 3 * 4$

$3 + -5 / 2 * 2$

$! == ! \quad || \quad 3 / 2 == 4$
 $\&\& \quad 2 == 4 / 2$

Precedence of Java Operators

Category	Operator	Associativity	Precedence
Postfix	() [] . (dot)	Left to right	Highest
Unary	++ -- ! ~	Right to left	
Multiplicative	* / %	Left to right	
Additive	+ -	Left to right	
Shift	>> >>> <<	Left to right	
Relational	> >= < <=	Left to right	
Equality	== !=	Left to right	
Bitwise AND	&	Left to right	
Bitwise XOR	^	Left to right	
Bitwise OR		Left to right	
Logical AND	&&	Left to right	
Logical OR		Left to right	
Conditional	?:	Right to left	
Assignment	= += -= *= /=	Right to left	
Comma	,	Left to right	Lowest

$2 * 3 / 2$ 3

$3 / 2 * 2$ 2

$2 + 3 / 3 * 4$ 6

$3 + -5 / 2 * 2$ -1

$1 == 1 || 3 / 2 == 4$
 $\&\& 2 == 4 / 2$ TRUE

While Loop

```
while(Boolean_expression)
{
    //Statements
}
```

The statements inside {} keep executing until this statement is true

```
public class HelloWorld {

    public static void main(String args[]) {
        int x = 10;
        while (x < 20) {
            System.out.print("value of x : " + x);
            x++;
            System.out.print("\n");
        }
    }
}
```


While Loop

```
while(Boolean_expression)
{
    //Statements
}
```

The statements inside {} keep executing until this statement is true

```
public class HelloWorld {
    public static void main(String args[]) {
        int x = 10;
        while (x < 20) {
            System.out.print("value of x : " + x);
            x++;
            System.out.print("\n");
        }
    }
}
```

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```


do ... while Loop

```
do
{
    //Statements
}while(Boolean_expression);
```

The statements inside {} keep executing until this statement is true

```
public class HelloWorld {

    public static void main(String args[]) {
        int x = 10;
        do {
            System.out.print("value of x : " + x);
            x++;
            System.out.print("\n");
        } while (x < 20);
    }
}
```


do ... while Loop

```
do
{
    //Statements
}while(Boolean_expression);
```

The statements inside {} keep executing until this statement is true

```
public class HelloWorld {

    public static void main(String args[]) {
        int x = 10;
        do {
            System.out.print("value of x : " + x);
            x++;
            System.out.print("\n");
        } while (x < 11);
    }
}
```

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```


for Loop

```
for(initialization; Boolean_expression; update)
{
    //Statements
}
```

1

Executed only once in beginning

2

5

...

Keeps running till this condition is true

4

7

...

Executed after the each execution of block

```
public class HelloWorld {
    public static void main(String args[]) {
        for(int x = 10; x < 20; x++) {
            System.out.print("value of x : " + x);
            x++;
            System.out.print("\n");
        }
    }
}
```

3

6

...

```
value of x : 10
value of x : 12
value of x : 14
value of x : 16
value of x : 18
```


Enhanced for Loop

```
for(declaration : expression)
{
    //Statements
}
```

```
public class HelloWorld {
    public static void main(String args[]){
        int [] numbers = {10, 20, 30, 40, 50};

        for(int x : numbers ){
            System.out.print( x );
            System.out.print(",");
        }

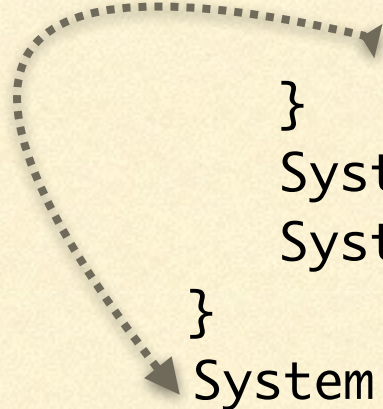
        System.out.print("\n");
        String [] names ={"James", "Larry", "Tom", "Lacy"};
        for( String name : names ) {
            System.out.print( name );
            System.out.print(",");
        }
    }
}
```

```
10,20,30,40,50,
James,Larry,Tom,Lacy,
```


Break Statement

Breaks the loop

```
public class Test {  
  
    public static void main(String args[]) {  
        int [] numbers = {10, 20, 30, 40, 50};  
  
        for(int x : numbers ) {  
            if( x == 30 ) {  
                break;  
            }  
            System.out.print( x );  
            System.out.print("\n");  
        }  
        System.out.print("Finished");  
    }  
}
```



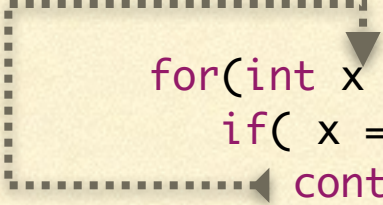
Output:

10
20

Continue Statement

Skips current flow and moves to beginning

```
public class Test {  
  
    public static void main(String args[]) {  
        int [] numbers = {10, 20, 30, 40, 50};  
  
        for(int x : numbers ) {  
            if( x == 30 ) {  
                continue;  
            }  
            System.out.print( x );  
            System.out.print("\n");  
        }  
    }  
}
```



Output:

10
20
40
50

Decision Making: If Statement

```
if(Boolean_expression)
{
    //Statements will execute if the Boolean expression is true
}
```

```
public class Test {

    public static void main(String args[]){
        int x = 10;

        if( x < 20 ){
            System.out.print("This is if statement");
        }
    }
}
```

Output:
This is if statement

Decision Making:The if...else Statement

```
if(Boolean_expression){  
    //Executes when the Boolean expression is true  
}else{  
    //Executes when the Boolean expression is false  
}
```

```
public class Test {  
  
    public static void main(String args[]){  
        int x = 30;  
  
        if( x < 20 ){  
            System.out.print("This is if statement");  
        }else{  
            System.out.print("This is else statement");  
        }  
    }  
}
```

Output:

This is else statement

Decision Making: The if...else if...else

```
if(Boolean_expression 1){  
    //Executes when the Boolean expression 1 is true  
}else if(Boolean_expression 2){  
    //Executes when the Boolean expression 2 is true  
}else {  
    //Executes when the none of the above condition is true.  
}
```

```
public class Test {  
    public static void main(String args[]){  
        int x = 30;  
        if( x == 10 ){  
            System.out.print("Value of X is 10");  
        }else if( x == 20 ){  
            System.out.print("Value of X is 20");  
        }else if( x == 30 ){  
            System.out.print("Value of X is 30");  
        }else{  
            System.out.print("This is else statement");  
        }  
    }  
}
```

Output:
Value of X is 30

Decision Making: Switch

```
switch(expression){  
    case value :  
        //Statements  
        break; //optional  
    case value :  
        //Statements  
        break; //optional  
    //You can have any  
    //number of case statements.  
    default : //Optional  
        //Statements  
}
```

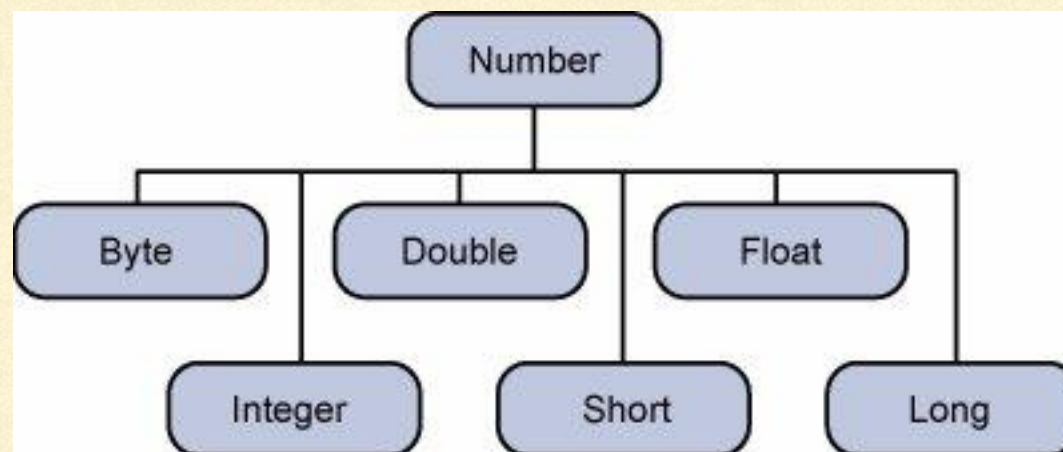
```
public class Test {  
    public static void main(String args[]){  
        char grade = 'C';  
        switch(grade)  
        {  
            case 'A' :  
                System.out.println("Excellent!");  
                break;  
            case 'B' :  
            case 'C' :  
                System.out.println("Well done");  
                break;  
            case 'D' :  
                System.out.println("You passed");  
            case 'F' :  
                System.out.println("Better try again");  
                break;  
            default :  
                System.out.println("Invalid grade");  
        }  
        System.out.println("Your grade is " + grade);  
    }  
}
```

Output:

```
Well done  
Your grade is C
```


Numbers

The primitive types are enough a number of time.
There are wrapper classes for primitive types.



```
public class Test{  
  
    public static void main(String args[]){  
        Integer x = 5; // boxes int to an Integer object  
        x = x + 10;    // unboxes the Integer to a int  
        System.out.println(x);  
    }  
}
```


Strings

```
public class Test{
    public static void main(String args[]){
        char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.' };
        String helloString = new String(helloArray);
        System.out.println( helloString ); //hello.

        String palindrome = "saw_I_was";
        System.out.println("Length: " + palindrome.length()); //Length: 9

        System.out.println(palindrome.concat("!!!")); //saw_I_was!!!
        System.out.println(palindrome); //saw_I_was

        System.out.println(palindrome.indexOf('D')); //-1

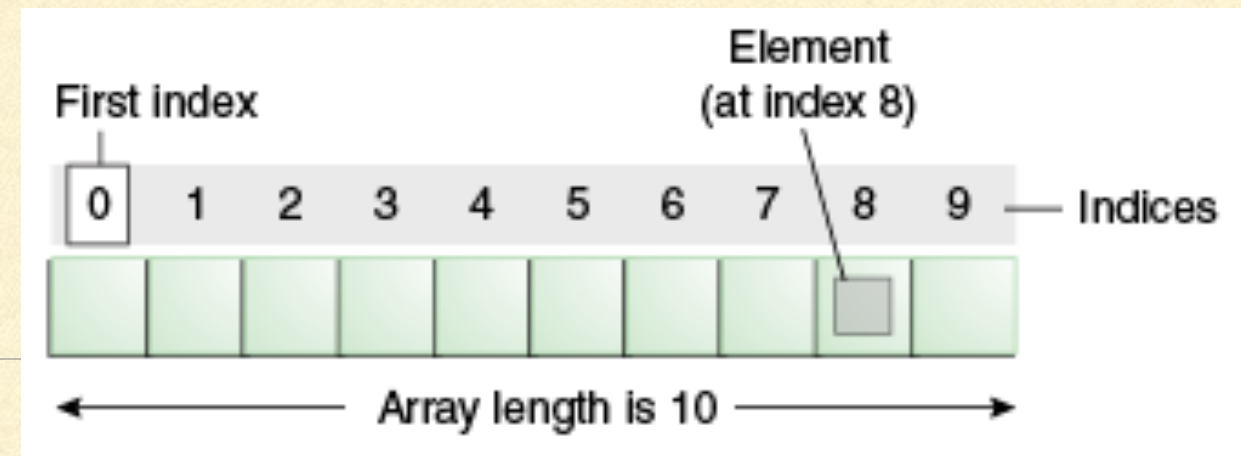
        System.out.println(palindrome.toUpperCase()); //SAW_I_WAS
        System.out.println(palindrome.replaceAll("[AEIOU]", "_")); //saw__was
        System.out.println(palindrome.startsWith("saw")); //true

        System.out.println(palindrome.substring(2)); //w_I_was
        System.out.println(palindrome.substring(1,2)); //a
    }
}
```

<http://docs.oracle.com/javase/7/docs/api/java/lang/String.html?is-external=true>

Arrays: A container object that holds fixed number of values of single type.

```
public class ArrayDemo {  
    public static void main(String[] args) {  
        // declares an array of integers  
        int[] anArray;  
  
        // allocates memory for 3 integers  
        anArray = new int[3];  
  
        // initialize first element  
        anArray[0] = 100;  
        // initialize second element  
        anArray[1] = 200;  
        // and so forth  
        anArray[2] = 300;  
  
        System.out.println("Element at index 0: " + anArray[0]);  
        System.out.println("Element at index 1: " + anArray[1]);  
        System.out.println("Element at index 2: " + anArray[2]);  
    }  
}
```



```
Element at index 0: 100  
Element at index 1: 200  
Element at index 2: 300
```

<http://docs.oracle.com/javase/7/docs/api/java/lang/String.html?is-external=true>

Arrays Operations

```
import java.util.Arrays;

public class ArrayDemo {
    public static void main(String[] args) {
        int[] myarr = {1,4, 5,6,10,3, 2};
        Arrays.sort(myarr);
        System.out.println(Arrays.toString(myarr));//[1, 2, 3, 4, 5, 6, 10]
        System.out.println(Arrays.binarySearch(myarr, 10));//6

        //returns - (insert point + 1)
        System.out.println(Arrays.binarySearch(myarr, 9)); //-7,

        Arrays.fill(myarr, 2);
        System.out.println(Arrays.toString(myarr));//[2, 2, 2, 2, 2, 2, 2]
    }
}
```

You could also try `copyOf()` and other functions.

<http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html>

Date Time

```
import java.util.Date;


public class DateDemo {
    public static void main(String args[]) {
        // Instantiate a Date object
        Date date = new Date();

        // display time and date using toString()
        System.out.println(date.toString());
    }
}
```

Output: 
Mon Feb 16 18:39:58 IST 2015

```
import java.util.*;
import java.text.*;

public class DateDemo {
    public static void main(String args[]) {
        Date dNow = new Date( );
        SimpleDateFormat ft =
            new SimpleDateFormat (
                "E yyyy.MM.dd 'at' hh:mm:ss a zzz");
        System.out.println("Current Date: "
            + ft.format(dNow));
    }
}
```

Output: 
Current Date: Mon 2015.02.16 at 06:40:56 PM IST

<http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html>

Methods - Creating Method

Collection of statement Grouped Together to perform operation

Modifier

Return Value

Name of the function

Arguments

Statements
Defining the
body.
Should have
"return"

```
public static int funcName(int a, int b) {  
    // body  
}
```


Methods - Creating Method

Collection of statement Grouped Together to perform operation

```
public class Test
{
    /** the snippet returns the minimum between two
    numbers */
    public static int minFunction(int n1, int n2) {
        int min;
        if (n1 > n2)
            min = n2;
        else
            min = n1;
        return min;
    }
    public static void main( String args[] ){
        System.out.println(minFunction(10, 20));
    }
}
```

10

Methods - Creating Void Returning Method

```
public class ExampleVoid {  
  
    public static void main(String[] args) {  
        methodRankPoints(255.7);  
    }  
  
    public static void methodRankPoints(double points) {  
        if (points >= 202.5) {  
            System.out.println("Rank:A1");  
        }  
        else if (points >= 122.4) {  
            System.out.println("Rank:A2");  
        }  
        else {  
            System.out.println("Rank:A3");  
        }  
    }  
}
```

Rank:A1

Methods - Passing Parameters by Value:

```
public class swappingExample {  
  
    public static void main(String[] args) {  
        int a = 30;  
        int b = 45;  
  
        System.out.println("Before swapping, a = " +  
                            a + " and b = " + b);  
        // Invoke the swap method  
        swapFunction(a, b);  
        System.out.println("After swapping outside, a = " +  
                            a + " and b is " + b);  
    }  
  
    public static void swapFunction(int a, int b) {  
  
        System.out.println("Before swapping(Inside), a = " + a  
                            + " b = " + b);  
        // Swap n1 with n2  
        int c = a;  
        a = b;  
        b = c;  
        System.out.println("After swapping(Inside), a = " + a  
                            + " b = " + b);  
    }  
}
```

Before swapping, a = 30 and b = 45
Before swapping(Inside), a = 30 b = 45
After swapping(Inside), a = 45 b = 30
After swapping outside, a = 30 and b is 45

Methods Overloading

```
public class ExampleOverloading {
    public static void main(String[] args) {
        int a = 11;
        int b = 6;
        double c = 7.3;
        double d = 9.4;
        int result1 = minFunction(a, b);
        // same function name with different parameters
        double result2 = minFunction(c, d);
        System.out.println("Minimum Value = " + result1);
        System.out.println("Minimum Value = " + result2);
    }
    public static int minFunction(int n1, int n2) {
        int min;
        if (n1 > n2)
            min = n2;
        else
            min = n1;
        return min;
    }
    public static double minFunction(double n1, double n2) {
        double min;
        if (n1 > n2)
            min = n2;
        else
            min = n1;
        return min;
    }
}
```

Minimum Value = 6
Minimum Value = 7.3

Java - Streams, Files and I/O - Byte Streams

```
import java.io.*;

public class CopyFile {
    public static void main(String args[]) throws IOException
    {
        FileInputStream in = null;
        FileOutputStream out = null;

        try {
            in = new FileInputStream("input.txt");
            out = new FileOutputStream("output.txt");

            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }
}
```

Minimum Value = 6
Minimum Value = 7.3

Java - Streams, Files and I/O - Standard Streams

```
import java.io.*;

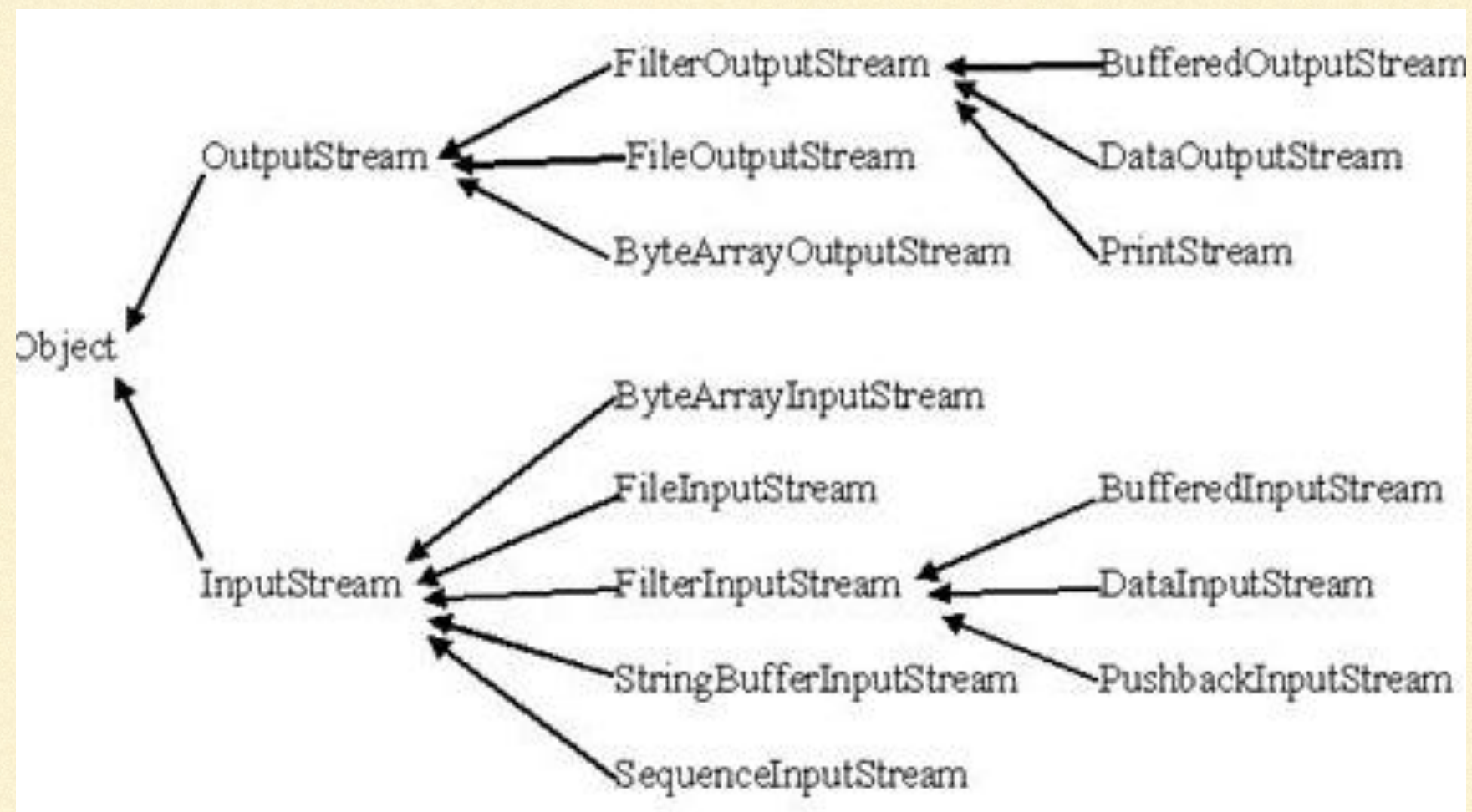
public class ReadConsole {
    public static void main(String args[]) throws IOException
    {
        InputStreamReader cin = null;

        try {
            cin = new InputStreamReader(System.in);
            System.out.println("Enter characters, 'q' to quit.");
            char c;
            do {
                c = (char) cin.read();
                System.out.print(c);
            } while(c != 'q');
        } finally {
            if (cin != null) {
                cin.close();
            }
        }
    }
}
```

Enter characters, 'q' to quit.

hi
hi
hi
q
q

Java - Streams, Files and I/O - Library



Java - Streams, Files and I/O - File Reading

```
import java.io.*;

public class FileStreamTest{

    public static void main(String args[]){

        try{
            byte bWrite [] = {11,21,3,40,5};
            OutputStream os = new FileOutputStream("test.txt");
            for(int x=0; x < bWrite.length ; x++){
                os.write( bWrite[x] ); // writes the bytes
            }
            os.close();

            InputStream is = new FileInputStream("test.txt");
            int size = is.available();

            for(int i=0; i< size; i++){
                System.out.print((char)is.read() + " ");
            }
            is.close();
        }catch(IOException e){
            System.out.print("Exception");
        }
    }
}
```


Java - Streams, Files and I/O - Directories

ls

```
import java.io.File;

public class ReadDir {
    public static void main(String[] args) {

        File file = null;
        String[] paths;

        try{
            // create new file object
            file = new File("/tmp");

            // array of files and directory
            paths = file.list();

            // for each name in the path array
            for(String path:paths)
            {
                // prints filename and directory name
                System.out.println(path);
            }
        }catch(Exception e){
            // if any error occurs
            e.printStackTrace();
        }
    }
}
```

mkdir

```
import java.io.File;

public class CreateDir {
    public static void main(String args[]) {
        String dirname = "/tmp/user/java/bin";
        File d = new File(dirname);
        // Create directory now.
        d.mkdirs();
    }
}
```



Java - Exception

```
import java.io.*;
public class ExcepTest{

    public static void main(String args[]){
        try{
            int a[] = new int[2];
            System.out.println("Access element three :" + a[3]);
        }catch(ArrayIndexOutOfBoundsException e){
            System.out.println("Exception thrown  :" + e);
        }
        System.out.println("Out of the block");
    }
}
```

Throw your own exceptions

```
import java.io.*;
public class className
{
    public void deposit(double amount) throws RemoteException
    {
        // Method implementation
        throw new RemoteException();
    }
    //Remainder of class definition
}
```

```
try
{
    //Protected code
}catch(ExceptionType1 e1)
{
    //Catch block
}catch(ExceptionType2 e2)
{
    //Catch block
}catch(ExceptionType3 e3)
{
    //Catch block
}finally
{
    //The finally block always
    executes.
}
```


Java - Exception

```
import java.io.*;

public class InsufficientFundsException
extends Exception {
    private double amount;

    public InsufficientFundsException(double amount) {
        this.amount = amount;
    }

    public double getAmount() {
        return amount;
    }
}
```

```
import java.io.*;

public class CheckingAccount {
    private double balance;
    private int number;

    public CheckingAccount(int number) {
        this.number = number;
    }

    public void deposit(double amount) {
        balance += amount;
    }

    public void withdraw(double amount)
        throws InsufficientFundsException {
        if (amount <= balance) {
            balance -= amount;
        } else {
            double needs = amount - balance;
            throw new InsufficientFundsException(needs);
        }
    }

    public double getBalance() {
        return balance;
    }

    public int getNumber() {
        return number;
    }
}
```


Java - Exception

```
import java.io.*;

public class InsufficientFundsException
extends Exception {
    private double amount;

    public InsufficientFundsException(double amount) {
        this.amount = amount;
    }

    public double getAmount() {
        return amount;
    }
}
```

```
public class BankDemo
{
    public static void main(String [] args)
    {
        CheckingAccount c = new CheckingAccount(101);
        System.out.println("Depositing $500...");
        c.deposit(500.00);
        try
        {
            System.out.println("\nWithdrawing $100...");
            c.withdraw(100.00);
            System.out.println("\nWithdrawing $600...");
            c.withdraw(600.00);
        } catch (InsufficientFundsException e)
        {
            System.out.println("Sorry, but you are short $"
                               + e.getAmount());
            e.printStackTrace();
        }
    }
}
```

```
import java.io.*;

public class CheckingAccount {
    private double balance;
    private int number;

    public CheckingAccount(int number) {
        this.number = number;
    }

    public void deposit(double amount) {
        balance += amount;
    }

    public void withdraw(double amount)
        throws InsufficientFundsException {
        if (amount <= balance) {
            balance -= amount;
        } else {
            double needs = amount - balance;
            throw new InsufficientFundsException(needs);
        }
    }

    public double getBalance() {
        return balance;
    }

    public int getNumber() {
        return number;
    }
}
```



OOP - Inheritance - An object acquires properties of another

```
public class Animal
{
    int weight;
    Animal(int wt){weight = wt;}
    public void move(){
        System.out.println("I can move.!");
    }
    public void eat(String stuff)
    {
        System.out.println("yum yum..."
            + stuff);
    }
}
```

```
public class Dog extends Animal {
    String myName;
    private Dog(name, wt){
        super(wt); myName=name;
    }
    public Dog(String name) {
        this.myName = name;
    }
    public void bark(String atWho) {
        System.out.println("I'm "
            + myName + ", " + atWho);
    }
}
```

```
public class Main {

    public static void main(String[] args) {
        Dog tommy = new Dog("Tommy");
        tommy.move();
        tommy.eat("bread");
        tommy.bark("sandeep");
    }
}
```

I can move.!
yum yum...bread
I'm Tommy,sandeep

- Dog is a subclass of Animal
- Animal is called super class
- Subclass can't access private members
- protected used by self or subclass
- A final class can't be extended
- Done by extends & Implements
- Use the super's constructor by super()

OOP - Inheritance - Notes

- Every class extends Object
- Every class has:
 - hashCode()
 - Used while preparing hashset etc.
 - toString()
 - Converts to string
 - compareTo()
 - how does it compare to another object. $a > b$
 - equals()
 - Is this object equal to another.

```
public class Dog {
    String myName;
    Dog(String name){
        myName=name;
    }
    public String toString()
    {
        return myName;
    }
    public boolean equals(Dog d)
    {
        return d.myName == this.myName;
    }
    public static void main(String[] args) {
        Dog d = new Dog("h");
        System.out.println(d);
        Dog d1 = new Dog("h");
        System.out.println(d.equals(d1));
    }
}
```

h
true

IS-A & HAS-A relationship

```
public class Vehicle
{
    int weight;
    public void move(){
        ...
    }
}
```

```
public class Car extends Vehicle {
    Steering steering;
    Gear gear;
    ...
}
```

HAS-A relationship:
Car has steering

IS-A Relationship:
Car is a Vehicle

OOP - Encapsulation - Ability to hide internals

```
public class Vehicle
{
    private Wheel frontWL,
        rearWL, frontWR, rearWR;
    public void move()
    {
        frontWL.rotateACW();
        frontWR.rotateACW();
        rearWL.rotateACW();
        rearWR.rotateACW();
    }
}
```

- Helps making field readonly
 - provide only getters
- Behaviour is hidden from user

OOP - Polymorphism - An object takes multiple forms

```
public class Animal
{
    int weight;
    public void move(){
        System.out.println("I can move.!");
    }
    public void eat(String stuff)
    {
        System.out.println("yum yum..."
            + stuff);
    }
}
```

```
public class Dog extends Animal {

    String myName;

    public Dog(String name) {
        this.myName = name;
    }

    public void bark(String atWho) {
        System.out.println("I am "
            + myName + ". bho bho " + atWho);
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Animal pet = new Dog("Tommy");
        pet.move();
        pet.eat("bread");

        ((Dog)pet).bark("sandeep");
    }
}
```

I can move.!
yum yum...bread
I am Tommy. bho bho sandeep

- pet is a Dog
- pet is an animal
- Can assign subclass obj to super class
- Can't assign superclass obj to subclass

OOP - Overriding - An object takes multiple forms

```
public class Animal
{
    int weight;
    public void move(){
        System.out.println("I can move.!");
    }
    public void eat(String stuff)
    {
        System.out.println("yum yum..."
            + stuff);
    }
}
```

```
public class Dog extends Animal {
    String myName;
    public Dog(String name) {
        this.myName = name;
    }
    public void eat(String stuff)
    {
        System.out.println("yikes: "
            + stuff);
    }
    public void bark(String atWho) {
        System.out.println("I am "
            + myName + ". bho bho " + atWho);
    }
}
```

```
public class Main {

    public static void main(String[] args) {
        Animal pet = new Dog("Tommy");
        pet.move();
        pet.eat("bread");

        Animal pet2 = new Animal();
        pet2.eat("bread");
    }
}
```

I can move.!
yikes: bread
yum yum bread.

OOP - Abstract Class - A class for which you can't create objects

- Same as normal class defined with a keyword "abstract"
- Does not have constructor
- Can not create instance
- May have abstract methods

```
abstract class GraphicObject {  
    int x, y;  
  
    void moveTo(int newX, int newY) {  
        x = newX + 10;  
        y = newY + 10;  
    }  
    abstract void draw();  
    abstract void resize();  
}
```

```
abstract class GraphicObject {  
    int x, y;  
  
    void moveTo(int newX, int newY) {  
        x = newX + 10;  
        y = newY + 10;  
    }  
}
```

```
✗ GraphicObject go = new GraphicObject();
```


OOP - Interface - A blue print or a contract

- Is basically a contract
- Collection of Abstract methods
- Has methods
- Does not have constructors
- Does not have any member variables
- Name of file should be same as name of interface
- has package
- Class implement all methods
 - or make it abstract class

```
public interface Creature {  
    public void move();  
    public void eat(String stuff);  
}
```

```
public class Animal implements Creature  
{  
    int weight;  
    public void move(){  
        System.out.println("I can move.!");  
    }  
    public void eat(String stuff)  
    {  
        System.out.println("yum yum..." + stuff);  
    }  
}
```

Interface

What's wrong with this?

```
public interface Creature {  
    public void move();  
    public void eat(String stuff);  
}
```

```
public class Animal implements Creature  
{  
    int weight;  
    public void move(){  
        System.out.println("I can move.!");  
    }  
}
```

Interface

What's wrong with this?

```
public interface Creature {  
    public void move();  
    public void eat(String stuff);  
}
```

```
public class Animal implements Creature  
{  
    int weight;  
    public void move(){  
        System.out.println("I can move.!");  
    }  
}
```

The eat() function is not implemented.

Interface

What's wrong with this?

```
public interface Creature {  
    public void move();  
    public void eat();  
}
```

```
public class Animal implements Creature  
{  
    int weight;  
    public void move(){  
        System.out.println("I can move.!");  
    }  
    public void eat(String stuff)  
    {  
        System.out.println("yum yum..." + stuff);  
    }  
}
```

Interface

What's wrong with this?

```
public interface Creature {  
    public void move();  
    public void eat();  
}
```

```
public class Animal implements Creature  
{  
    int weight;  
    public void move(){  
        System.out.println("I can move.!");  
    }  
    public void eat(String stuff)  
    {  
        System.out.println("yum yum..." + stuff);  
    }  
}
```

The eat() has different signature.

Data Structures - BitSet - Group of bits or flags

```
public class HelloWorld {  
    public static void main(String args[]) {  
        BitSet bits1 = new BitSet(16);  
        BitSet bits2 = new BitSet(16);  
        // set some bits  
        for(int i=0; i<16; i++) {  
            if((i%2) == 0) bits1.set(i);  
            if((i%5) != 0) bits2.set(i);  
        }  
        System.out.println("Initial pattern in bits1: ");  
        System.out.println(bits1);  
        System.out.println("\nInitial pattern in bits2: ");  
        System.out.println(bits2);  
  
        // AND bits  
        bits2.and(bits1);  
        System.out.println("\nbits2 AND bits1: ");  
        System.out.println(bits2);  
  
        // OR bits  
        bits2.or(bits1);  
        System.out.println("\nbits2 OR bits1: ");  
        System.out.println(bits2);  
  
        // XOR bits  
        bits2.xor(bits1);  
        System.out.println("\nbits2 XOR bits1: ");  
        System.out.println(bits2);  
    }  
}
```

Initial pattern in bits1:
{0, 2, 4, 6, 8, 10, 12, 14}

Initial pattern in bits2:
{1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14}

bits2 AND bits1:
{2, 4, 6, 8, 12, 14}

bits2 OR bits1:
{0, 2, 4, 6, 8, 10, 12, 14}

bits2 XOR bits1:
{}



Data Structures - Vector - Array that grows.

Can have Different Types of Elements.

```
import java.util.*;
public class VectorDemo {

    public static void main(String args[]) {
        // initial size is 3, increment is 2
        Vector v = new Vector(3, 2);
        System.out.println("Initial size: " + v.size());
        System.out.println("Initial capacity: " + v.capacity());
        v.addElement(new Integer(1));
        v.addElement(new Float(2.3));
        v.addElement(new Double(6.08));
        System.out.println("Current capacity: " + v.capacity());

        System.out.println("First: " + (Integer) v.firstElement());
        System.out.println("Last: " + (Double) v.lastElement());
        if (v.contains(new Integer(3)))
            System.out.println("Vector contains 3.");

        // enumerate the elements in the vector.
        Enumeration vEnum = v.elements();
        System.out.println("\nElements in vector:");

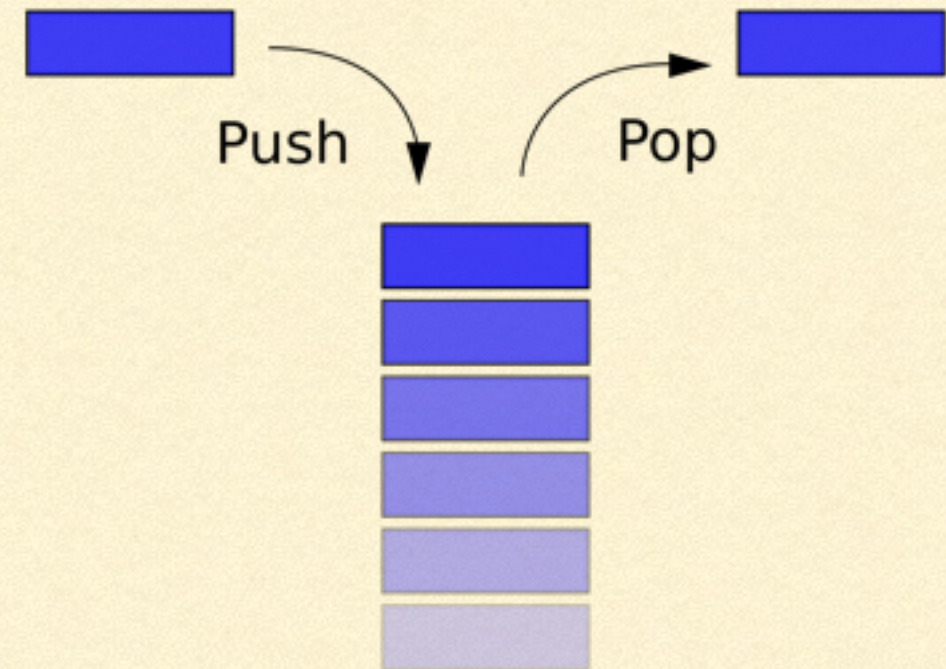
        while (vEnum.hasMoreElements())
            System.out.print(vEnum.nextElement() + " ");
        System.out.println();
    }
}
```

```
Initial size: 0
Initial capacity: 3
Current capacity: 3
First element: 1
Last element: 6.08
```

```
Elements in vector:
1 2.3 6.08
```


Data Structures - Stack - Gives most recently inserted

```
import java.util.*;
public class StackDemo {
    static void showpush(Stack st, int a) {
        st.push(new Integer(a));
        System.out.println("push(" + a + ")");
        System.out.println("stack: " + st);
    }
    static void showpop(Stack st) {
        System.out.print("pop -> ");
        Integer a = (Integer) st.pop();
        System.out.println(a);
        System.out.println("stack: " + st);
    }
    public static void main(String args[]) {
        Stack st = new Stack();
        System.out.println("stack: " + st);
        showpush(st, 42);
        showpush(st, 66);
        showpop(st);
        showpop(st);
        try {
            showpop(st);
        } catch (EmptyStackException e) {
            System.out.println("empty stack");
        }
    }
}
```



```
stack: []
push(42)
stack: [42]
push(66)
stack: [42, 66]
pop -> 66
stack: [42]
pop -> 42
stack: []
pop -> empty stack
```


Data Structures - Hashtable - Bunch of key-values

```
public class HashtableDemo {  
  
    public static void main(String args[]) {  
        // Create a hash map  
        Hashtable balance = new Hashtable();  
        String str;  
        double bal;  
  
        balance.put("Zara", new Double(3434.34));  
        balance.put("Mahnaz", new Double(123.22));  
  
        // Show all balances in hash table.  
        Enumeration names = balance.keys();  
        while (names.hasMoreElements()) {  
            str = (String) names.nextElement();  
            System.out.println(str + ": " + balance.get(str));  
        }  
  
        // Deposit 1,000 into Zara's account  
        bal = ((Double) balance.get("Zara")).doubleValue();  
        balance.put("Zara", new Double(bal + 1000));  
        System.out.println("Zara's new balance: "  
            + balance.get("Zara"));  
    }  
}
```

Zara: 3434.34
Mahnaz: 123.22
Zara's new balance: 4434.34

Data Structures - Properties - Managing Config key-values

```
public class PropDemo {  
  
    public static void main(String args[]) {  
        Properties capitals = new Properties();  
        Set states;  
        String str;  
  
        capitals.put("Illinois", "Springfield");  
        capitals.put("Missouri", "Jefferson City");  
        capitals.put("Washington", "Olympia");  
        capitals.put("California", "Sacramento");  
        capitals.put("Indiana", "Indianapolis");  
  
        // Show all states and capitals in hashtable.  
        states = capitals.keySet(); // get set-view of keys  
        Iterator itr = states.iterator();  
        while(itr.hasNext()) {  
            str = (String) itr.next();  
            System.out.println("The capital of " +  
                str + " is " + capitals.getProperty(str) + ".");  
        }  
        System.out.println();  
  
        // look for state not in list -- specify default  
        str = capitals.getProperty("Florida", "Not Found");  
        System.out.println("The capital of Florida is "  
            + str + ".");  
    }  
}
```

The capital of Missouri is Jefferson City.
The capital of Illinois is Springfield.
The capital of Indiana is Indianapolis.
The capital of California is Sacramento.
The capital of Washington is Olympia.

The capital of Florida is Not Found.

- Used for managing the key-values
- System.getProperties() uses it
- It is possible to chain

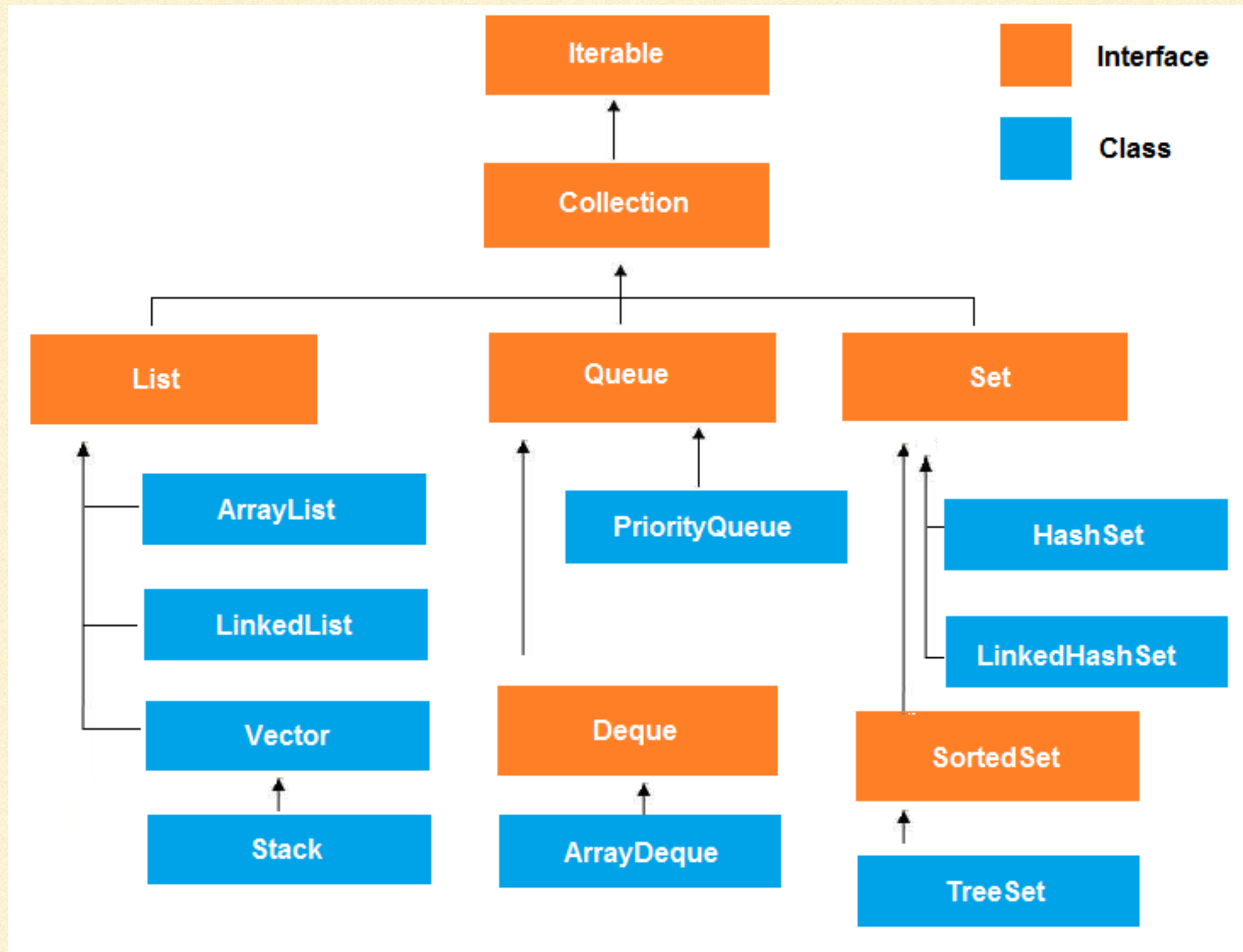
Collections

- Library of Data Structures
 - Dynamic arrays - ArrayList
 - Linked lists - LinkedList
 - Trees
 - Hashtables
- Highly efficient.
- High degree of interoperability
- Extending and/or adapting easy

Categorised:

- Interfaces
- Implementations
- Algorithms e.g. Sorting, Searching

Collections



Data Structures - ArrayList

```
public class ArrayListDemo {  
    public static void main(String args[]) {  
        // create an array list  
        List al = new ArrayList();  
        System.out.println("Initial size of al: " + al.size());  
  
        // add elements to the array list  
        al.add("C");  
        al.add("A");  
        al.add("F");  
        al.add(1, "A2");  
        System.out.println("Size of al after additions: " + al.size());  
  
        // display the array list  
        System.out.println("Contents of al: " + al);  
        // Remove elements from the array list  
        al.remove("F");  
        al.remove(2);  
        System.out.println("Size of al after deletions: " + al.size());  
        System.out.println("Contents of al: " + al);  
    }  
}
```

Initial size of al: 0
Size of al after additions: 4
Contents of al: [C, A2, A, F]
Size of al after deletions: 2
Contents of al: [C, A2]

Collections - Collection Algorithms

Defines several algorithms that can be applied to collections.

```
public class AlgorithmsDemo {  
    public static void main(String args[]) {  
        LinkedList ll = new LinkedList();  
        ll.add(new Integer(-8));  
        ll.add(new Integer(20));  
        ll.add(new Integer(-20));  
        ll.add(new Integer(8));  
        // Create a reverse order comparator  
        Comparator r = Collections.reverseOrder();  
        // Sort list by using the comparator  
        Collections.sort(ll, r);  
        // Get iterator  
        Iterator li = ll.iterator();  
        System.out.print("List sorted in reverse: ");  
        while (li.hasNext())  
            System.out.print(li.next() + " ");  
  
        Collections.shuffle(ll);  
        // display randomized list  
        li = ll.iterator();  
        System.out.print("\nList shuffled: ");  
        while (li.hasNext())  
            System.out.print(li.next() + " ");  
  
        System.out.println("\nMinimum: " + Collections.min(ll));  
        System.out.println("Maximum: " + Collections.max(ll));  
    }  
}
```

List sorted in reverse: 20 8 -8 -20
List shuffled: 20 8 -20 -8
Minimum: -20
Maximum: 20

Generics - Write code for all datatypes

Imagine you are creating - reverse()

```
public class MyLibrary {  
  
    public void reverse(int [] arr)  
    {  
        int mid = arr.length/2;  
        for(int i=0; i<mid; i++)  
        {  
            int tmp = arr[i];  
            arr[i] = arr[arr.length-i-1];  
            arr[arr.length-i-1] = tmp;  
        }  
    }  
  
    public static void main(String[] args) {  
        MyLibrary ml = new MyLibrary();  
        int[] a = new int[]{1, 4, 5, 6};  
        ml.reverse(a);  
        System.out.println(Arrays.toString(a));  
    }  
}
```

What about other data types?

Generics - Write code for all datatypes

Imagine you are creating - reverse()

```
public class MyLibrary {  
  
    public void reverse(Object [] arr)  
    {  
        int mid = arr.length/2;  
        for(int i=0; i<mid; i++)  
        {  
            Object tmp = arr[i];  
            arr[i] = arr[arr.length-1];  
            arr[arr.length-1] = tmp;  
        }  
    }  
  
    public static void main(String[] args) {  
        MyLibrary ml = new MyLibrary();  
        Object[] a = new Integer[]{1, 4, 5, 6};  
        ml.reverse(a);  
        System.out.println(Arrays.toString(a));  
    }  
}
```

Problems?

Generics - Write code for all datatypes

Imagine you are creating - reverse()

```
public class MyLibrary {  
  
    public void reverse(Object [] arr)  
    {  
        int mid = arr.length/2;  
        for(int i=0; i<mid; i++)  
        {  
            Object tmp = arr[i];  
            arr[i] = arr[arr.length-1];  
            arr[arr.length-1] = tmp;  
        }  
    }  
  
    public static void main(String[] args) {  
        MyLibrary ml = new MyLibrary();  
        Object[] a = new Integer[]{1, 4, 5, 6};  
        ml.reverse(a);  
        System.out.println(Arrays.toString(a));  
    }  
}
```

Problems?

It is no longer type safe.

Generics - Write code for all datatypes

Imagine you are creating - reverse()

```
import java.util.*;

public class MyLibrary<T> {

    public void reverse(T [] arr)
    {
        int mid = arr.length/2;
        for(int i=0; i<mid; i++)
        {
            T tmp = arr[i];
            arr[i] = arr[arr.length-1];
            arr[arr.length-1] = tmp;
        }
    }

    public static void main(String[] args) {
        MyLibrary<Integer> ml = new MyLibrary<Integer>();
        Integer[] a = new Integer[]{1, 4, 5, 6};
        ml.reverse(a);
        System.out.println(Arrays.toString(a));
    }
}
```

Generics

While extending you could define the datatypes.

```
public class StubMapper extends Mapper<Object, Text, IntWritable,
LongWritable> {

    @Override
    public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException {

        String line = value.toString();

        for(int i=0; i < line.length(); i++)
        {
            char ch = line.charAt(i);
            context.write(new IntWritable(ch), new LongWritable(1));
        }
    }
}
```


Generics

While extending you could define the datatypes.

```
public class Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {  
  
    protected void map(KEYIN key, VALUEIN value, Context context)  
        throws IOException, InterruptedException {  
        context.write((KEYOUT) key, (VALUEOUT) value);  
    }  
  
}
```

```
public class StubMapper extends Mapper<Object, Text, IntWritable,  
LongWritable> {  
    @Override  
    public void map(Object key, Text value, Context context)  
        throws IOException, InterruptedException {  
        String line = value.toString();  
        for(int i=0; i < line.length(); i++)  
        {  
            char ch = line.charAt(i);  
            context.write(new IntWritable(ch), new LongWritable(1));  
        }  
    }  
}
```




Thank you for attending
Java Basics

JavaTM



+91-9538998962

sandeep@knowbigdata.com