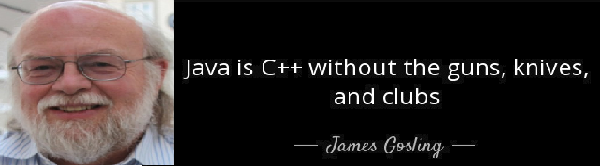
**KGiSL**

***We make IT happen***

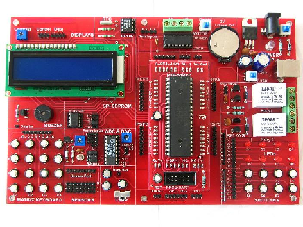
**BOOTCAMP II – Tutorial**

******

***Core JAVA – I am everwhere………….***

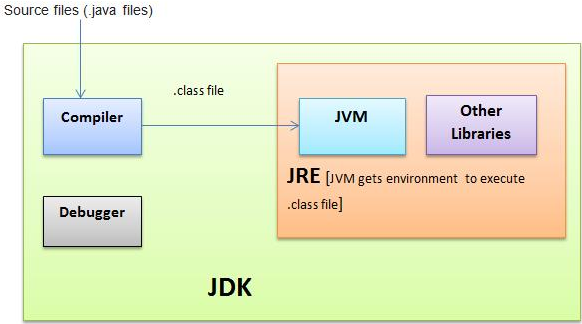
******

***JDK- Java Development Kit***

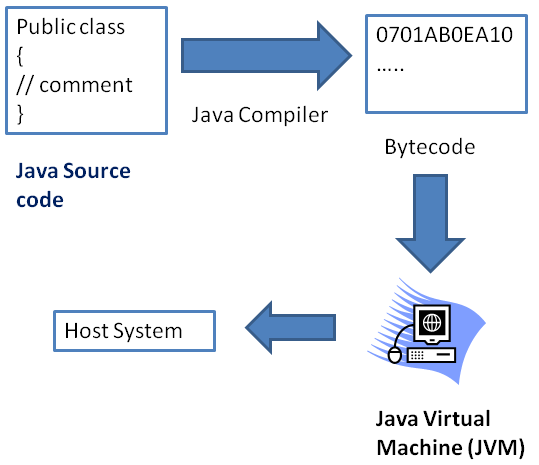
******



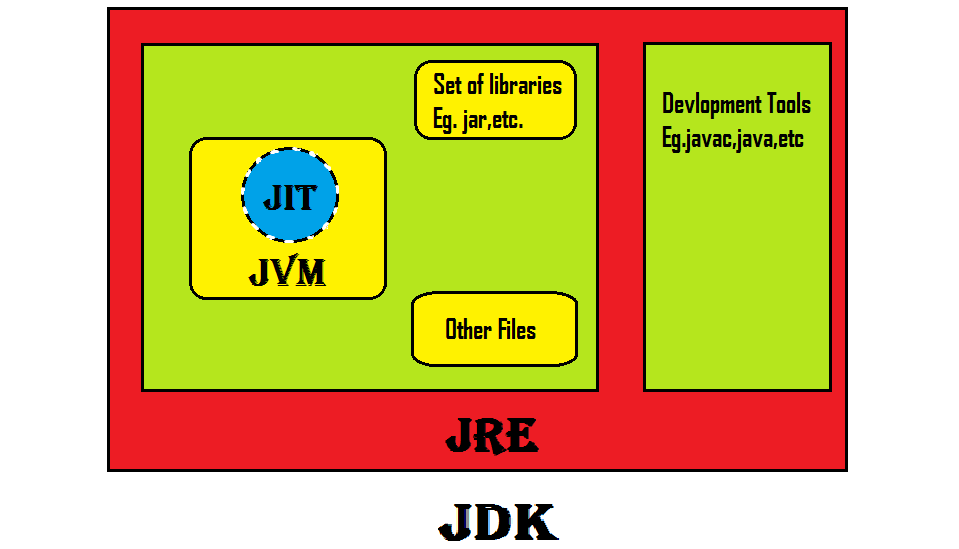
The JDK includes a private JVM and a few other resources to finish the development of a Java Application. Since the introduction of the [Java](https://en.wikipedia.org/wiki/Java_(software_platform)) platform, it has been by far the most widely used Software Development Kit ([SDK](https://en.wikipedia.org/wiki/Software_development_kit)).

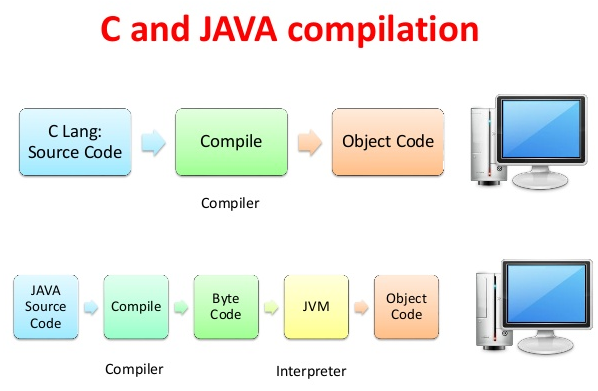
******

***JVM***

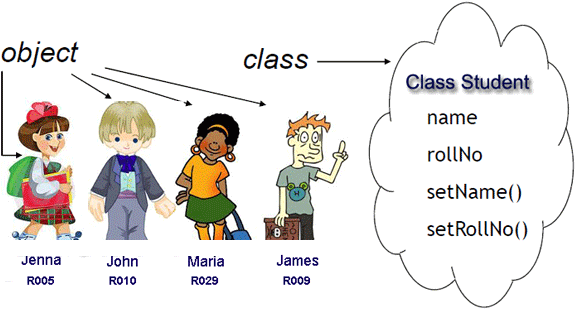
******

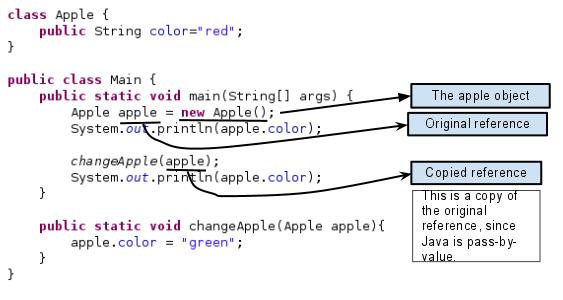
***JRE***

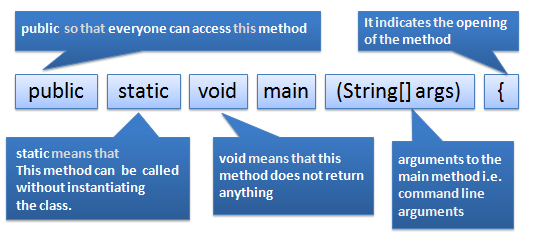


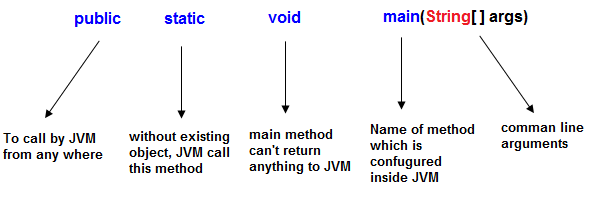
******

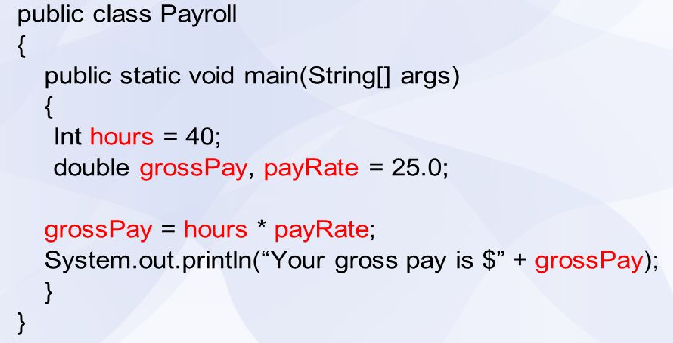
***Classes, Objects, Methods***

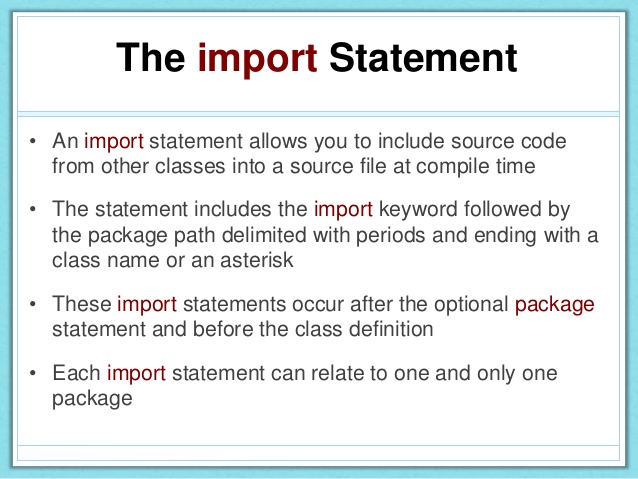
******

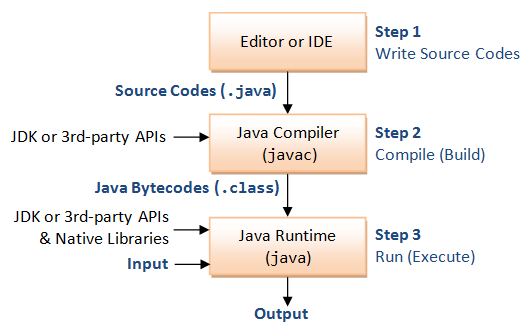




******

****





**Example:**

**import** java.util.Scanner;

**class** AddNumbers

{

**public** **static** **void** main(String args[])

{

**int** x, y, z;

System.out.println("Enter two integers to calculate their sum ");

Scanner in = **new** Scanner(System.in);

x = in.nextInt();

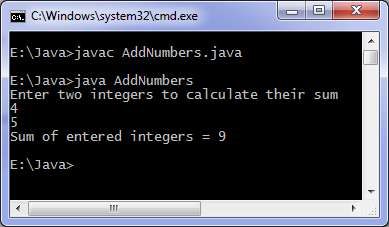
y = in.nextInt();

z = x + y;

System.out.println("Sum of entered integers = "+z);

}

}

**Output of program:**  


Above code can add only numbers in range of integers(4 bytes), if you wish to add very large numbers then you can use BigInteger class.

**Code to add very large numbers:**

**import** java.util.Scanner;

**import** java.math.BigInteger;

**class** AddingLargeNumbers {

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

String number1, number2;

Scanner in = **new** Scanner(System.in);

System.out.println("Enter first large number");

number1 = in.nextLine();

System.out.println("Enter second large number");

number2 = in.nextLine();

BigInteger first = **new** BigInteger(number1);

BigInteger second = **new** BigInteger(number2);

BigInteger sum;

sum = first.add(second);

System.out.println("Result of addition = " + sum);

}

}

In our code we create two objects of BigInteger class in java.math package. Input should be digit strings otherwise an exception will be raised, also you cannot simply use '+' operator to add objects of BigInteger class, you have to use add method for addition of two objects.

**Output of program:**

Enter first large number

11111111111111

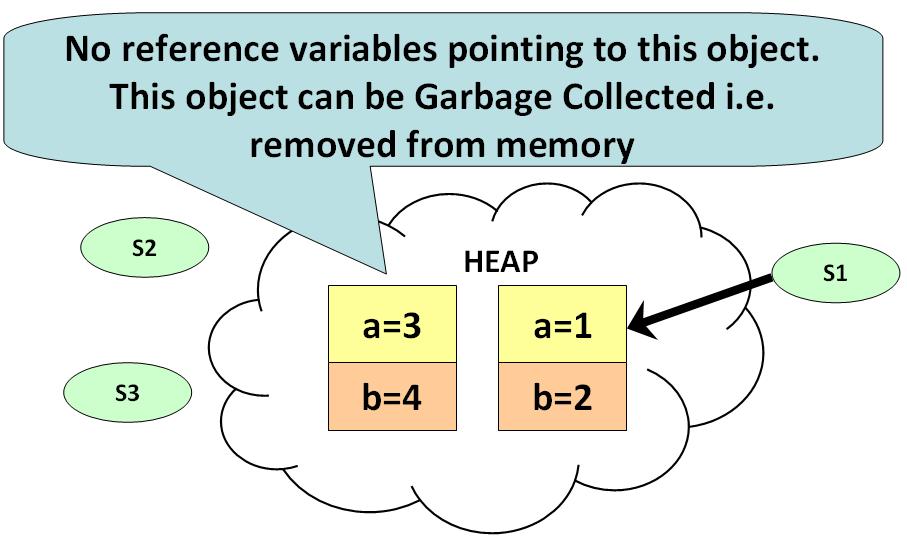
Enter second large number

99999999999999

Result of addition = 111111111111110

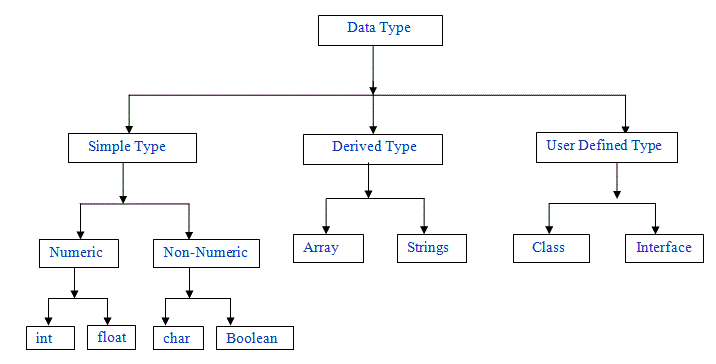
***Garbage Collector***

******

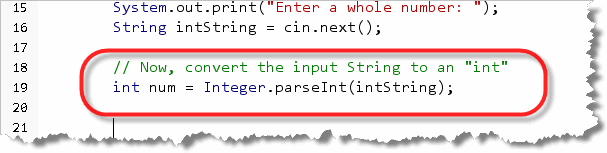


* Garbage means unreferenced objects.
* Garbage Collection is process of reclaiming the runtime unused memory automatically. It is a way to destroy the unused objects.
* We were using free () function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.
* It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.
* It is automatically done by the garbage collector (a part of JVM).
* All primitive data types in Java are signed. Java does not support unsigned types.
* Local variables – **Stack** Memory Dynamic Objects – **Heap** Memory
* Now, an object can also be declared on the heap. For the sake of this discussion, think of the heap as an amorphous blob of memory. Unlike the stack, which automatically allocates and de-allocates the necessary memory as you enter and exit stack frames, you must manually reserve and free heap memory.
* An object declared on the heap does, after a fashion, "survive" between stack frames. One could say that an object declared on the heap never goes out of scope, but that's really because the object is never really associated with any scope. Such an object must be created via the new keyword, and must be referred to by a pointer.
* It is your responsibility to free the heap object once you are done with it. You free heap objects with the delete keyword. The destructor on a heap object is not called until you free the object.
* The pointers that refer to heap objects are themselves usually local variables associated with scopes. Once you are done using the heap object, you allow the pointer(s) referring to it to go out of scope. If you haven't explicitly freed the object the pointer is pointing to, then the block of heap memory will never be freed until the process exits (this is called a memory leak).
* Think of it all this way: an object created on the stack is like a balloon taped to a chair in a room. When you exit the room, the balloon automatically pops. An object created on the heap is like a balloon on a ribbon, tied to a chair in a room. The ribbon is the pointer. When you exit the room, the ribbon automatically vanishes, but the balloon just floats to the ceiling and takes up space. The proper procedure is to pop the balloon with a pin, and then exit the room, whereupon the ribbon will disappear. But, the good thing about the balloon on the string is you can also untie the ribbon, hold it in your hand, and exit the room and take the balloon with you.
* So to go to your linked list example: typically, nodes of such a list are declared on the heap, with each node holding a pointer to the next node. All of this is sitting on the heap and never goes out of scope. The only thing that could go out of scope is the pointer that points to the root of the list - the pointer you use to reference into the list in the first place. That can go out of scope.

***Datatypes***



***parseInt()***



***In-built classes (commomly used)***

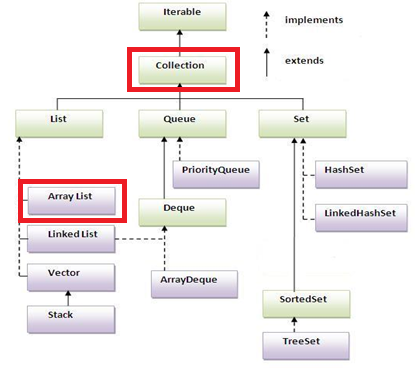
1. **java.lang.String**  
   [String](http://javapapers.com/core-java/java-string/) class will be the undisputed champion on any day by popularity and none will deny that. This is a final class and used to create / operate immutable string literals. It was available from JDK 1.0
2. **java.lang.System**  
   Usage of [System](http://javapapers.com/core-java/system-out-println/) depends on the type of project you work on. You may not be using it in your project but still it is one of the popular java classes around. This is a utility class and cannot be instantiated. Main uses of this class are access to standard input, output, environment variables, etc. Available since JDK 1.0
3. **java.lang.Exception**  
   Throwable is the super class of all Errors and Exceptions. All abnormal conditions that can be handled comes under Exception. [NullPointerException](http://javapapers.com/core-java/nullpointerexception-null-bad-good-and-ugly/) is the most popular among all the exceptions. Exception is at top of hierarchy of all such exceptions. Available since JDK 1.0
4. **java.util.ArrayList**  
   An implementation of array data structure. This class implements List interface and is the most popular member or java collections framework. Difference between ArrayList and Vector is one popular topic among the beginners and frequently asked question in java interviews. It was introduced in JDK 1.2
5. **java.util.HashMap**  
   An implementation of a key-value pair data structure. This class implements Map interface. As similar to ArrayList vs Vector, we have HashMap vs [Hashtable](http://javapapers.com/core-java/java-hashtable/" \o "Java Hashtable)popular comparisons. This happens to be a popular collection class that acts as a container for property-value pairs and works as a transport agent between multiple layers of an application. It was introduced in JDK 1.2.
6. **java.lang.Object**  
   Great grandfather of all java classes. Every java class is a subclass of Object. It will be used often when we work on a platform/framework. It contains the important methods like equals, hashcode, clone, toString, etc. It is available from day one of java (JDK 1.0)
7. **java.lang.Thread**  
   A thread is a single sequence of execution, where multiple thread can co-exist and share resources. We can extend this Thread class and create our own threads. Using Runnable is also another option. Usage of this class depends on the domain of your application. It is not absolutely necessary to build a usual application. It was available from JDK 1.0
8. **java.lang.Class**  
   Class is a direct subclass of Object. There is no constructor in this class and their objects are loaded in JVM by [classloaders](http://javapapers.com/core-java/java-class-loader/" \o "Java Class Loader). Most of us may not have used it directly but I think it is an essential class. It is an important class in doing reflection. It is available from JDK 1.0
9. **java.util.Date**  
   This is used to work with date. Sometimes we feel that this class should have added more utility methods and we end up creating those. Every enterprise application we create has a date utility. Introduced in JDK 1.0 and later made huge changes in JDK1.1 by deprecating a whole lot of methods.
10. **java.util.Iterator**  
    This is an interface. It is very popular and came as a replacement for Enumeration. It is a simple to use convenience utility and works in sync with Iterable. It was introduced in JDK 1.2

***Collections***

Collections in java is a framework that provides an architecture to store and manipulate the group of objects. All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections. Collection represents a single unit of objects i.e. a group. Collection framework represents a unified architecture for storing and manipulating group of objects. It has:

1. Interfaces and its implementations i.e. classes
2. Algorithm

The java.util package contains all the classes and interfaces for Collection framework.



***Methods of Collection interface***

There are many methods declared in the Collection interface. They are as follows:

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | public boolean add(Object element) | is used to insert an element in this collection. |
| 2 | public boolean addAll(collection c) | is used to insert the specified collection elements in the invoking collection. |
| 3 | public boolean remove(Object element) | is used to delete an element from this collection. |
| 4 | public boolean removeAll(Collection c) | is used to delete all the elements of specified collection from the invoking collection. |
| 5 | public boolean retainAll(Collection c) | is used to delete all the elements of invoking collection except the specified collection. |
| 6 | public int size() | return the total number of elements in the collection. |
| 7 | public void clear() | removes the total no of element from the collection. |
| 8 | public boolean contains(object element) | is used to search an element. |
| 9 | public boolean containsAll(Collection c) | is used to search the specified collection in this collection. |
| 10 | public Iterator iterator() | returns an iterator. |
| 11 | public Object[] toArray() | converts collection into array. |
| 12 | public boolean isEmpty() | checks if collection is empty. |
| 13 | public boolean equals(Object element) | matches two collection. |
| 14 | public int hashCode() | returns the hashcode number for collection. |

### *ArrayList Example:*

import java.util.\*;

public class ArrayListExample {

public static void main(String args[]) {

/\*Creation of ArrayList: I'm going to add String

\*elements so I made it of string type \*/

ArrayList<String> obj = new ArrayList<String>();

/\*This is how elements should be added to the array list\*/

obj.add("Ajeet");

obj.add("Harry");

obj.add("Chaitanya");

obj.add("Steve");

obj.add("Anuj");

/\* Displaying array list elements \*/

System.out.println("Currently the array list has following elements:"+obj);

/\*Add element at the given index\*/

obj.add(0, "Rahul");

obj.add(1, "Justin");

/\*Remove elements from array list like this\*/

obj.remove("Chaitanya");

obj.remove("Harry");

System.out.println("Current array list is:"+obj);

/\*Remove element from the given index\*/

obj.remove(1);

System.out.println("Current array list is:"+obj);

}

}

**Output:**

Currently the array list has following elements:[Ajeet, Harry, Chaitanya, Steve, Anuj]

Current array list is:[Rahul, Justin, Ajeet, Steve, Anuj]

Current array list is:[Rahul, Ajeet, Steve, Anuj]

***Practice with Coding Standards***

1. Write a program to ask the user to enter marks obtained in exam. Compare the input marks against minimum passing marks. Print the appropriate message on the screen based on whether user passed the exam or not.
2. Write a program to print multiplication table of a number entered by the user using a for loop. You can modify it for while or do while loop for practice.
3. Write a program to check whether a number entered by the user is Strong Harshad/Weak Harshad number.
4. Write a program to print Floyd’s triangle.
5. Explain the below code,

import java.util.\*;

class GarbageCollection

{

public static void main(String s[]) throws Exception

{

Runtime rs = Runtime.getRuntime();

System.out.println("Free memory in JVM before Garbage Collection = "+rs.freeMemory());

rs.gc();

System.out.println("Free memory in JVM after Garbage Collection = "+rs.freeMemory());

}

}

1. Write a program to reverse a number entered by the user. If the input is 951 then output will be 159.
2. Explain the code given below,

import java.util.\*;

import java.io.\*;

class Notepad {

public static void main(String[ ] args) {

Runtime rs = Runtime.getRuntime();

try {

rs.exec("notepad");

}

catch (IOException e) {

System.out.println(e);

}

}

}

1. Explain the code given below,

import java.net.InetAddress;

class IPAddress

{

public static void main(String args[]) throws Exception

{

System.out.println(InetAddress.getLocalHost());

}

}

1. Write a program to display date and time.
2. Write a program to compare 2 strings entered by the user.

***Interfaces***



It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface. Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.

***Example:***

interface Mobile {

public void call();

public void browse();

}

A class uses the **implements** keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

/\* File name : Samsung.java \*/

public class Samsung implements Mobile{

public void call(){

System.out.println("Call Activated");

}

public void browse(){

System.out.println("Search - Google");

}

public int storeImages(){

return 0;

}

public static void main(String args[]){

Samsung s = new Samsung();

s.call();

s.browse();

}

}

**Output:**

Call Activated

Search - Google

***Packages***

******

Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations easier, etc. Some of the existing packages in Java are::

* **java.lang** - bundles the fundamental classes
* **java.io** - classes for input , output functions are bundled in this package

Programmers can define their own packages to bundle group of classes/interfaces, etc. It is a good practice to group related classes implemented by you so that a programmer can easily determine that the classes, interfaces, enumerations, annotations are related.

***Example:***

package Mobiles;

/\* File name : Mobile.java \*/

interface Mobile {

public void call();

public void browse();

}

package Mobiles;

/\* File name : Samsung.java \*/

public class Samsung implements Mobile{

public void call(){

System.out.println("Call Activated");

}

public void browse(){

System.out.println("Search - Google");

}

public int storeImages(){

return 0;

}

public static void main(String args[]){

Samsung s = new Samsung();

s.call();

s.browse();

}

}

***Inheritance***

 ******

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. The keyword **extends** is used to inherit the properties of a class.

***Example:***

class Calculation{

int z;

public void addition(int x, int y){

z = x+y;

System.out.println("The sum of the given numbers:"+z);

}

public void Substraction(int x,int y){

z = x-y;

System.out.println("The difference between the given numbers:"+z);

}

}

public class My\_Calculation extends Calculation{

public void multiplication(int x, int y){

z = x\*y;

System.out.println("The product of the given numbers:"+z);

}

public static void main(String args[]){

int a = 20, b = 10;

My\_Calculation demo = new My\_Calculation();

demo.addition(a, b);

demo.Substraction(a, b);

demo.multiplication(a, b);

}

}

**Output:**

The sum of the given numbers:30

The difference between the given numbers:10

The product of the given numbers:200

***Applets***

An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

There are some important differences between an applet and a standalone Java application, including the following:

* An applet is a Java class that extends the java.applet.Applet class.
* A main() method is not invoked on an applet, and an applet class will not define main().
* Applets are designed to be embedded within an HTML page.
* When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
* A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
* The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
* Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.
* Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.

***Example***

import java.applet.\*;

import java.awt.\*;

public class HelloWorldApplet extends Applet

{

public void paint (Graphics g)

{

g.drawString ("Hello World", 25, 50);

}

}

***Invoking an Applet***

<html>

<title>The Hello, World Applet</title>

<hr>

<applet code="HelloWorldApplet.class" width="320" height="120">

If your browser was Java-enabled, a "Hello, World"

message would appear here.

</applet>

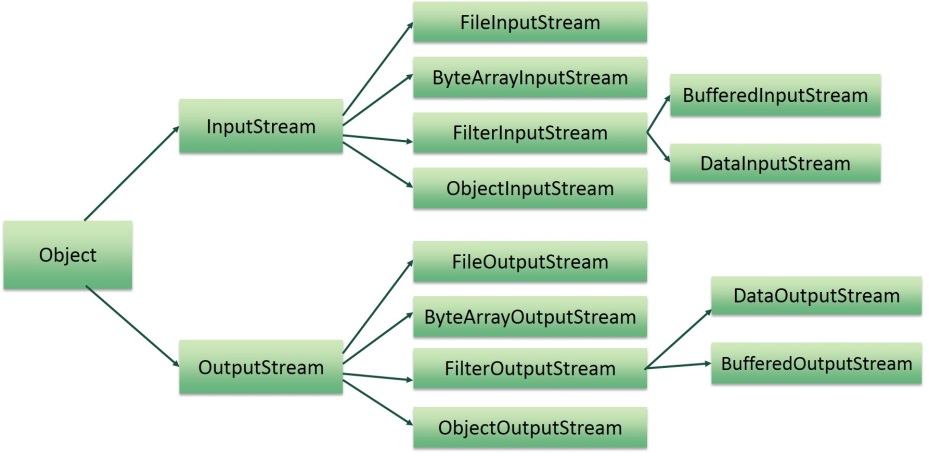
<hr>

</html>

***Practice with Coding Standards***

1. Find out duplicate number between 1 to N numbers.
2. Write a program to implement your own ArrayList class. It should contain add(), get(), remove(), size() methods. Use dynamic array logic. It should increase its size when it reaches threshold.
3. Write a program to implement hashcode and equals.
4. Write a program to swap or exchange two numbers. You should not use any temporary or third variable to swap.
5. Draw a rectangle in the browser screen using applets.
6. Program ATM machine using interfaces.
7. Program Store management system using inheritance and packages.

***Files & directories, Exception handling***



## FileInputStream:

This stream is used for reading data from the files. Objects can be created using the keyword new and there are several types of constructors available.

Following constructor takes a file name as a string to create an input stream object to read the file.:

InputStream f = new FileInputStream("C:/java/hello");

Following constructor takes a file object to create an input stream object to read the file. First we create a file object using File() method as follows:

File f = new File("C:/java/hello");

InputStream f = new FileInputStream(f);

Once you have *InputStream* object in hand, then there is a list of helper methods which can be used to read to stream or to do other operations on the stream.

|  |  |
| --- | --- |
| **SN** | **Methods with Description** |
| 1 | **public void close() throws IOException{}**  This method closes the file output stream. Releases any system resources associated with the file. Throws an IOException. |
| 2 | **protected void finalize()throws IOException {}**  This method cleans up the connection to the file. Ensures that the close method of this file output stream is called when there are no more references to this stream. Throws an IOException. |
| 3 | **public int read(int r)throws IOException{}**  This method reads the specified byte of data from the InputStream. Returns an int. Returns the next byte of data and -1 will be returned if it's end of file. |
| 4 | **public int read(byte[] r) throws IOException{}**  This method reads r.length bytes from the input stream into an array. Returns the total number of bytes read. If end of file -1 will be returned. |
| 5 | **public int available() throws IOException{}**  Gives the number of bytes that can be read from this file input stream. Returns an int. |

There are other important input streams available,

* [ByteArrayInputStream](http://www.tutorialspoint.com/java/java_bytearrayinputstream.htm)
* [DataInputStream](http://www.tutorialspoint.com/java/java_datainputstream.htm)

## FileOutputStream:

FileOutputStream is used to create a file and write data into it. The stream would create a file, if it doesn't already exist, before opening it for output.

Here are two constructors which can be used to create a FileOutputStream object.

Following constructor takes a file name as a string to create an input stream object to write the file:

OutputStream f = new FileOutputStream("C:/java/hello")

Following constructor takes a file object to create an output stream object to write the file. First, we create a file object using File() method as follows:

File f = new File("C:/java/hello");

OutputStream f = new FileOutputStream(f);

Once you have *OutputStream* object in hand, then there is a list of helper methods, which can be used to write to stream or to do other operations on the stream.

|  |  |
| --- | --- |
| **SN** | **Methods with Description** |
| 1 | **public void close() throws IOException{}**  This method closes the file output stream. Releases any system resources associated with the file. Throws an IOException |
| 2 | **protected void finalize()throws IOException {}**  This method cleans up the connection to the file. Ensures that the close method of this file output stream is called when there are no more references to this stream. Throws an IOException. |
| 3 | **public void write(int w)throws IOException{}**  This methods writes the specified byte to the output stream. |
| 4 | **public void write(byte[] w)**  Writes w.length bytes from the mentioned byte array to the OutputStream. |

There are other important output streams available, for more detail you can refer to the following links:

* [ByteArrayOutputStream](http://www.tutorialspoint.com/java/java_bytearrayoutputstream.htm)
* [DataOutputStream](http://www.tutorialspoint.com/java/java_dataoutputstream.htm)

## Example:

Following is the example to demonstrate InputStream and OutputStream:

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");

}

}

}

The above code would create file test.txt and would write given numbers in binary format. Same would be output on the stdout screen.

## File Navigation and I/O:

There are several other classes for File Navigation and I/O.

* [File Class](http://www.tutorialspoint.com/java/java_file_class.htm)
* [FileReader Class](http://www.tutorialspoint.com/java/java_filereader_class.htm)
* [FileWriter Class](http://www.tutorialspoint.com/java/java_filewriter_class.htm)

## *Directories in Java:*

A directory is a File which can contains a list of other files and directories. You use File object to create directories, to list down files available in a directory. For complete detail check a list of all the methods which you can call on File object and what are related to directories.

## Creating Directories:

There are two useful File utility methods, which can be used to create directories:

* The **mkdir( )** method creates a directory, returning true on success and false on failure. Failure indicates that the path specified in the File object already exists, or that the directory cannot be created because the entire path does not exist yet.
* The **mkdirs()** method creates both a directory and all the parents of the directory.

Following example creates "/tmp/user/java/bin" directory:

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();

}

}

Compile and execute above code to create "/tmp/user/java/bin".

**Note:** Java automatically takes care of path separators on UNIX and Windows as per conventions. If you use a forward slash (/) on a Windows version of Java, the path will still resolve correctly.

## Listing Directories:

You can use **list( )** method provided by **File** object to list down all the files and directories available in a directory as follows:

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();

}

}

}

This would produce following result based on the directories and files available in your **/tmp** directory:

test1.txt

test2.txt

ReadDir.java

ReadDir.class

***JDBC (overview)***

JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database. JDBC works with Java on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

*Note: This will be revisited in the next module.*

***Practice with Coding Standards***

# Write a program to list all files from a directory (or subdirectory)

1. Write a program to filter the files by file extensions and display the file names.
2. Write a program to read file content using byte array.
3. Write a program to read file content line by line.
4. Write a program to read input from java console.
5. Write a program to get file list from a folder filtered by extensions.
6. Write a program to get file URI reference.
7. Write a program to store and read objects from a file.
8. Write a program to get file last modified time.
9. Write a program to set file permissions.

***DBMS***



A database management system (DBMS) is a [computer software](https://en.wikipedia.org/wiki/Computer_software) application that interacts with the user, other applications, and the database itself to capture and analyze data. A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases. Well-known DBMSs include [MySQL](https://en.wikipedia.org/wiki/MySQL), [PostgreSQL](https://en.wikipedia.org/wiki/PostgreSQL), [Microsoft SQL Server](https://en.wikipedia.org/wiki/Microsoft_SQL_Server), [Oracle](https://en.wikipedia.org/wiki/Oracle_Database), [Sybase](https://en.wikipedia.org/wiki/Sybase) and [IBM DB2](https://en.wikipedia.org/wiki/IBM_DB2). A database is not generally [portable](https://en.wikipedia.org/wiki/Software_portability) across different DBMSs, but different DBMS can interoperate by using [standards](https://en.wikipedia.org/wiki/Technical_standard) such as [SQL](https://en.wikipedia.org/wiki/SQL) and [ODBC](https://en.wikipedia.org/wiki/ODBC) or [JDBC](https://en.wikipedia.org/wiki/JDBC) to allow a single application to work with more than one DBMS.

***SQL queries***



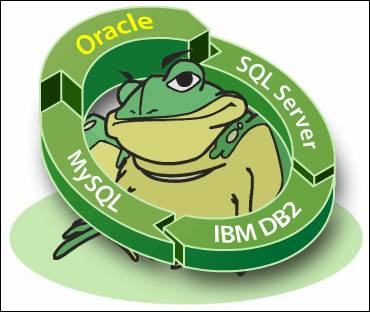
Structured Query Language is a [special-purpose programming language](https://en.wikipedia.org/wiki/Special-purpose_programming_language) designed for managing data held in a [relational database management system](https://en.wikipedia.org/wiki/Relational_database_management_system) (RDBMS), etc. The scope of SQL includes data insert, query, update and delete, [schema](https://en.wikipedia.org/wiki/Database_schema) creation and modification, and data access control.

***Oracle***



Oracle Database (commonly referred to as Oracle RDBMS or simply as Oracle) is an [object-relational database management system](https://en.wikipedia.org/wiki/Object-relational_database_management_system) produced and marketed by [Oracle Corporation](https://en.wikipedia.org/wiki/Oracle_Corporation).

***Introduction to TOAD***



Toad is a [software application](https://en.wikipedia.org/wiki/Software_application) from [Dell Software](https://en.wikipedia.org/wiki/Dell_Software) that [database](https://en.wikipedia.org/wiki/Database) developers, database administrators and data analysts use to manage both relational and non-relational databases using [SQL](https://en.wikipedia.org/wiki/SQL).

***Example:***

SELECT \* FROM Customers;

***Procedures***

DECLARE

a number;

b number;

c number;

PROCEDURE findMin(x IN number, y IN number, z OUT number) IS

BEGIN

IF x < y THEN

z:= x;

ELSE

z:= y;

END IF;

END;

BEGIN

a:= 23;

b:= 45;

findMin(a, b, c);

dbms\_output.put\_line(' Minimum of (23, 45) : ' || c);

END;

/

***JDBC (revisited)***

//STEP 1. Import required packages

import java.sql.\*;

public class FirstExample {

// JDBC driver name and database URL

static final String JDBC\_DRIVER = "com.mysql.jdbc.Driver";

static final String DB\_URL = "jdbc:mysql://localhost/EMP";

// Database credentials

static final String USER = "username";

static final String PASS = "password";

public static void main(String[] args) {

Connection conn = null;

Statement stmt = null;

try{

//STEP 2: Register JDBC driver

Class.forName("com.mysql.jdbc.Driver");

//STEP 3: Open a connection

System.out.println("Connecting to database...");

conn = DriverManager.getConnection(DB\_URL,USER,PASS);

//STEP 4: Execute a query

System.out.println("Creating statement...");

stmt = conn.createStatement();

String sql;

sql = "SELECT id, first, last, age FROM Employees";

ResultSet rs = stmt.executeQuery(sql);

//STEP 5: Extract data from result set

while(rs.next()){

//Retrieve by column name

int id = rs.getInt("id");

int age = rs.getInt("age");

String first = rs.getString("first");

String last = rs.getString("last");

//Display values

System.out.print("ID: " + id);

System.out.print(", Age: " + age);

System.out.print(", First: " + first);

System.out.println(", Last: " + last);

}

}

catch(Exception e){

//Handle errors for Class.forName

e.printStackTrace();

}finally{

// Clean-up environment

rs.close();

stmt.close();

conn.close();

}

System.out.println("Goodbye!");

} //end main

} //end FirstExample

**Output:**

Connecting to database...

Creating statement...

ID: 100, Age: 18, First: Zara, Last: Ali

ID: 101, Age: 25, First: Mahnaz, Last: Fatma

ID: 102, Age: 30, First: Zaid, Last: Khan

ID: 103, Age: 28, First: Sumit, Last: Mittal

**In Eclipse**

import java.sql.Connection;

import java.sql.DriverManager;

public class DBconn {

public static Connection getConnection() {

//Static : It means that this class is not instance related but class related. It can be accessed without creating the instance of Class.

//Initialize the connection variable

Connection con =null;

try{

//to dynamically load the driver's class file into memory, which automatically registers it

Class.forName("oracle.jdbc.driver.OracleDriver"); //to register the Oracle driver

//to establish a connection

//getConnection(String url, String user, String password)

//url - jdbc:oracle:thin:@hostname:port Number:databaseName

//The getConnection() method of DriverManager class is used to establish connection with the database

con= DriverManager.getConnection("jdbc:oracle:thin:@10.100.1.30:1521:kurnia","kurniadev","system");

}

catch (Exception e)

{

e.printStackTrace();

/\* With println: you only know what exception has been thrown

java.lang.UnsupportedOperationException: Not yet implemented

With printStackTrace: you also know what caused it (line numbers + call stack)

java.lang.UnsupportedOperationException: Not yet implemented

at javaapplication27.Test1.test(Test1.java:27)

at javaapplication27.Test1.main(Test1.java:19) \*/

}

return con;

}

}

***Practice with Coding Standards***

1. Create a bank application and store the transactions in the database. Allow the application to insert, delete, update, view the records.