Java

Data Types

Data types are divided into two groups:

- Primitive data types - includes byte, short, int, long, float, double, boolean and char
- Non-primitive data types - such as String, Arrays and Classes

VARIABLES IN JAVA

A Java variable is a piece of memory that can contain a data value. A variable thus has a data type. In Java there are four types of variables:

- Non-static fields
- Static fields
- Local variables
- Parameters

A non-static field is a variable that belongs to an object. Objects keep their internal state in non-static fields. Non-static fields are also called instance variables, because they belong to instances (objects) of a class. Non-static fields are covered in more detail in the text on **Java fields**.

A static field is a variable that belongs to a class. A static field has the same value for all objects that access it. Static fields are also called class variables. Static fields are also covered in more detail in the text on **Java fields**.

A local variable is a variable declared inside a method. A local variable is only accessible inside the method that declared it. Local variables are covered in more detail in the text on **Java methods**.

A parameter is a variable that is passed to a method when the method is called. Parameters are also only accessible inside the method that declares them, although a value is assigned to them when the method is called. Parameters are also covered in more detail in the text on **Java methods**.

VARIABLE TYPES

In Java there are four types of variables:

- Non-static fields
- Static fields
- Local variables
- Parameters

A non-static field is a variable that belongs to an object. Objects keep their internal state in non-static fields. Non-static fields are also called instance variables, because they belong to instances (objects) of a class. Non-static fields are covered in more detail in the text on **Java fields**.

A static field is a variable that belongs to a class. A static field has the same value for all objects that access it. Static fields are also called class variables. Static fields are also covered in more detail in the text on **Java fields**.

A local variable is a variable declared inside a method. A local variable is only accessible inside the method that declared it. Local variables are covered in more detail in the text on **Java methods**.

A parameter is a variable that is passed to a method when the method is called. Parameters are also only accessible inside the method that declares them, although a value is assigned to them when the method is called. Parameters are also covered in more detail in the text on **Java methods**.

JDK VS JRE VS JVM

1. JDK

Java Development Kit aka JDK is the core component of Java Environment and provides all the tools, executables, and binaries required to compile, debug, and execute a Java Program.

JDK is a platform-specific software and that’s why we have separate installers for Windows, Mac, and Unix systems.

We can say that JDK is the superset of JRE since it contains JRE with Java compiler, debugger, and core classes.

JVM

JVM is the heart of Java programming language. When we execute a Java program, JVM is responsible for converting the byte code to the machine-specific code.

JVM is also platform-dependent and provides core java functions such as memory management, garbage collection, security, etc.

JVM is customizable and we can use java options to customize it. For example, allocating minimum and maximum memory to JVM.

JVM is called **virtual** because it provides an interface that does not depend on the underlying operating system and machine hardware.

This independence from hardware and the operating system makes java program write-once-run-anywhere.

JRE

JRE is the implementation of JVM. It provides a platform to execute java programs. JRE consists of JVM, Java binaries, and other classes to execute any program successfully.

JRE doesn’t contain any development tools such as Java compiler, debugger, JShell, etc.

If you just want to execute a java program, you can install only JRE. You don’t need JDK because there is no development or compilation of java source code is required.

Now that we have a basic understanding of JDK, JVM, and JRE, let’s look into the difference between them.

Let’s look at some of the important differences between JDK, JRE, and JVM.

1. JDK is for development purpose whereas JRE is for running the java programs.
2. JDK and JRE both contains JVM so that we can run our java program.
3. JVM is the heart of java programming language and provides platform independence.

Loops

FOR Loop

The Java for loop is a control flow statement that iterates a part of the programs multiple times.

WHEN TO USE FOR LOOP

If the number of iteration is fixed, it is recommended to use for loop.

SYNTAX:

```
for(init;condition;incr/decr){

// code to be executed

}
```

WHILE

The Java while loop is a control flow statement that executes a part of the programs repeatedly on the basis of given boolean condition.

WHEN TO USE WHILE LOOP

If the number of iteration is not fixed, it is recommended to use while loop.

SYNTAX

```
while(condition){

//code to be executed

}
```

DO-WHILE

The Java do while loop is a control flow statement that executes a part of the programs at least once and the further execution depends upon the given boolean condition.

WHEN TO USE DO WHILE

If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop.

SYNTAX

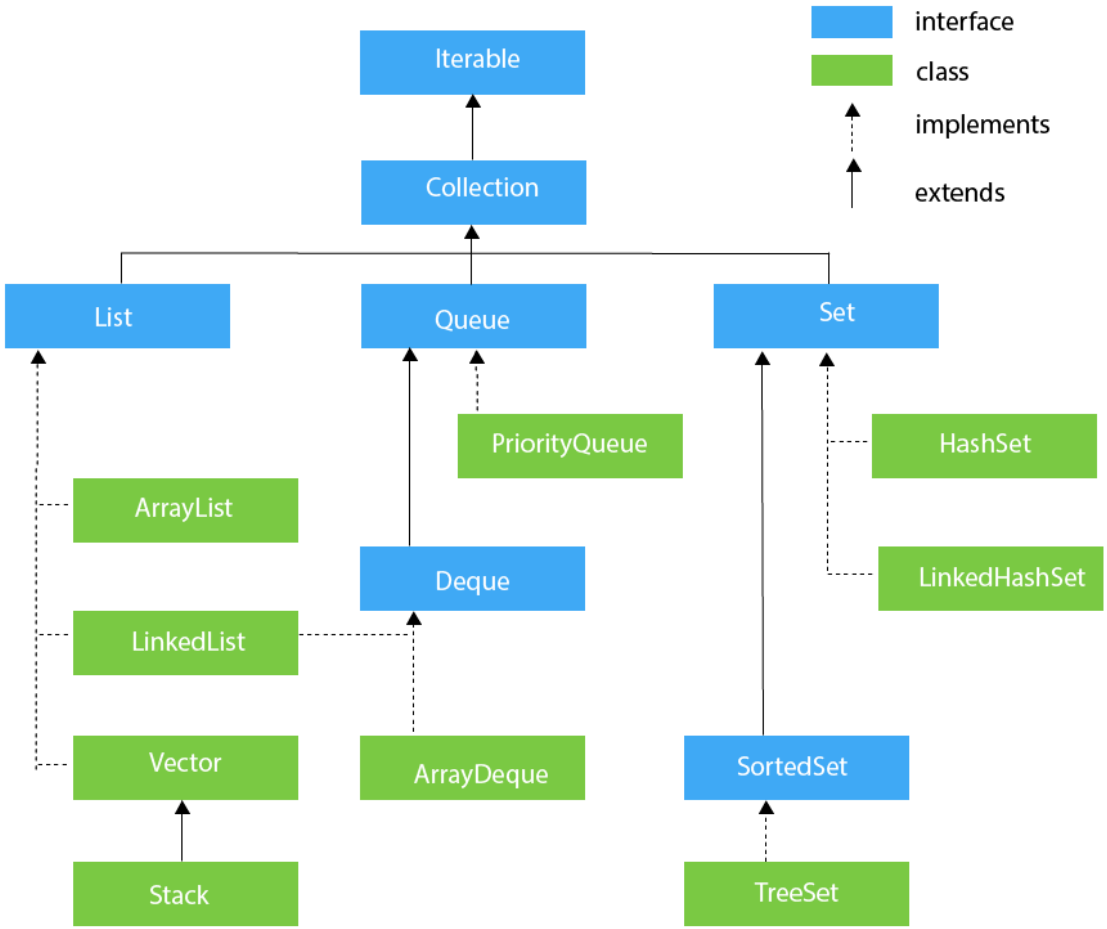
```
do{

//code to be executed

}while(condition);
```

Java Collections

Collection Framework is a combination of classes and interface, which is used to store and manipulate the data in the form of objects. It provides various classes such as ArrayList, Vector, Stack, and HashSet, etc. and interfaces such as List, Queue, Set, etc. for this purpose.

Hierarchy of Collection Framework

Let us see the hierarchy of Collection framework. The **java.util** package contains all the classes and interfaces for the Collection framework.



List Interface declaration

public interface List<E> extends Collection<E>

Creating a List of type String using ArrayList

List<String> list=new ArrayList<String>();

Creating a List of type Integer using ArrayList

List<Integer> list=new ArrayList<Integer>();

Creating a List of type Book using ArrayList

List<Book> list=new ArrayList<Book>();

Creating a List of type String using LinkedList

List<String> list=new LinkedList<String>();

Java ArrayList

Java ArrayList class uses a dynamic array for storing the elements. It is like an array, but there is no size limit. We can add or remove elements anytime. So, it is much more flexible than the traditional array. It is found in the java.util package.

The ArrayList in Java can have the duplicate elements also. It implements the List interface so we can use all the methods of List interface here. The ArrayList maintains the insertion order internally.

It inherits the AbstractList class and implements List interface.

ArrayList class declaration

public class ArrayList<E> extends AbstractList<E> implements List<E>

Constructor	Description
ArrayList()	It is used to build an empty array list.
ArrayList(Collection<? extends E> c)	It is used to build an array list that is initialized with the elements of the collection c.
ArrayList(int capacity)	It is used to build an array list that has the specified initial capacity.

Java Non-generic Vs. Generic Collection

non-generic example of creating java collection.

ArrayList list=new ArrayList();//creating old non-generic arraylist

generic example of creating java collection.

ArrayList<String> list=new ArrayList<String>();//creating new generic arraylist

Java LinkedList class

Java LinkedList class uses a doubly linked list to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.

The important points about Java LinkedList are:

Java LinkedList class can contain duplicate elements.

Java LinkedList class maintains insertion order.

Java LinkedList class is non synchronized.

In Java LinkedList class, manipulation is fast because no shifting needs to occur.

Java LinkedList class can be used as a list, stack or queue.

Vector

Vector uses a dynamic array to store the data elements. It is similar to ArrayList.
However, It is synchronized and contains many methods that are not the part of Collection framework.

Example: “ Vector<String> v=new Vector<String>();
v.add(“1");
v.add(“2");
Iterator<String> itr=v.iterator();
while(itr.hasNext())
{
System.out.println(itr.next());
} “

Stack

The stack is the subclass of Vector.
It implements the last-in-first-out data structure, i.e., Stack.
The stack contains all of the methods of Vector class and also provides its methods like boolean push(), boolean peek(), boolean push(object o), which defines its properties.

Example:

```
“Stack<String> stack = new Stack<String>();
 stack.push(“first element");
 stack.push(“second element");
 stack.pop();
 Iterator<String> itr=stack.iterator();
  while(itr.hasNext()){
 System.out.println(itr.next()); } “
```

HashSet

- HashSet class implements Set Interface.
- It represents the collection that uses a hash table for storage.
- Hashing is used to store the elements in the HashSet. It contains unique items.

Comparator

Comparator interface is used to order the objects of user-defined classes. A comparator object is capable of comparing two objects of two different classes. Following function compare obj1 with obj2.

Syntax:

```
public int compare(Object obj1, Object obj2):
```

How does Collections.Sort() work?

Internally the Sort method does call Compare method of the classes it is sorting. To compare two elements, it asks “Which is greater?” Compare method returns -1, 0 or 1 to say if it is less than, equal, or greater to the other. It uses this result to then determine if they should be swapped for its sort.

You can implement the Comparator method on a class in order to enable alternative sorting or searching methods through the java library.

For instance, if you wanted to sort an ArrayList of type String based on String length, then you could write a class called StringLengthComparator that implements Comparator with the desired comparison.

StringLengthComparator slc = new StringLengthComparator();

int result = slc.compare(a, b);

If a is shorter than b, then the method would return a number less than zero. If it is longer than b, then it would return a number greater than zero. If they are the same length, then it would return zero.
You could then sort the ArrayList using Collections.sort.

Collections.sort(myArrayList, slc);