Arrays Class from java.util package:

Arrays.copyOf : to copy array and increase/decrease size of newarr dynamically

Arrays.copyOfRange: to copy from index to To index.(here also we can change size)

Arrays.deepEquals(): compares two arrays and confirms equality

Arrays.equals():

Arrays.fill(): filling default values into all indexes

References:

<http://www.javawithus.com/tutorial/>

# Abstract Methods and Classes

An *abstract class* is a class that is declared abstract—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed.

An *abstract method* is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

abstract void moveTo(double deltaX, double deltaY);

If a class includes abstract methods, then the class itself *must* be declared abstract, as in:

public abstract class GraphicObject {

// declare fields

// declare nonabstract methods

abstract void draw();

}

When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared abstract.

**Note:** Methods in an *interface* (see the [Interfaces](https://docs.oracle.com/javase/tutorial/java/IandI/createinterface.html) section) that are not declared as default or static are *implicitly* abstract, so the abstract modifier is not used with interface methods. (It can be used, but it is unnecessary.)

## Abstract Classes Compared to Interfaces

Abstract classes are similar to interfaces. You cannot instantiate them, and they may contain a mix of methods declared with or without an implementation. However, with abstract classes, you can declare fields that are not static and final, and define public, protected, and private concrete methods. With interfaces, all fields are automatically public, static, and final, and all methods that you declare or define (as default methods) are public. In addition, you can extend only one class, whether or not it is abstract, whereas you can implement any number of interfaces.

Which should you use, abstract classes or interfaces?

* Consider using abstract classes if any of these statements apply to your situation:
  + You want to share code among several closely related classes.
  + You expect that classes that extend your abstract class have many common methods or fields, or require access modifiers other than public (such as protected and private).
  + You want to declare non-static or non-final fields. This enables you to define methods that can access and modify the state of the object to which they belong.
* Consider using interfaces if any of these statements apply to your situation:
  + You expect that unrelated classes would implement your interface. For example, the interfaces [Comparable](https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html) and [Cloneable](https://docs.oracle.com/javase/8/docs/api/java/lang/Cloneable.html" \t "_blank) are implemented by many unrelated classes.
  + You want to specify the behavior of a particular data type, but not concerned about who implements its behavior.
  + You want to take advantage of multiple inheritance of type.

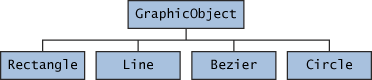
An example of an abstract class in the JDK is [AbstractMap](https://docs.oracle.com/javase/8/docs/api/java/util/AbstractMap.html" \t "_blank), which is part of the Collections Framework. Its subclasses (which include HashMap, TreeMap, and ConcurrentHashMap) share many methods (including get, put, isEmpty, containsKey, and containsValue) that AbstractMap defines.

An example of a class in the JDK that implements several interfaces is [HashMap](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html" \t "_blank), which implements the interfaces Serializable, Cloneable, and Map<K, V>. By reading this list of interfaces, you can infer that an instance of HashMap (regardless of the developer or company who implemented the class) can be cloned, is serializable (which means that it can be converted into a byte stream; see the section [Serializable Objects](https://docs.oracle.com/javase/tutorial/jndi/objects/serial.html" \t "_top)), and has the functionality of a map. In addition, the Map<K, V> interface has been enhanced with many default methods such as merge and forEach that older classes that have implemented this interface do not have to define.

Note that many software libraries use both abstract classes and interfaces; the HashMap class implements several interfaces and also extends the abstract class AbstractMap.

## An Abstract Class Example

In an object-oriented drawing application, you can draw circles, rectangles, lines, Bezier curves, and many other graphic objects. These objects all have certain states (for example: position, orientation, line color, fill color) and behaviors (for example: moveTo, rotate, resize, draw) in common. Some of these states and behaviors are the same for all graphic objects (for example: position, fill color, and moveTo). Others require different implementations (for example, resize or draw). All GraphicObjects must be able to draw or resize themselves; they just differ in how they do it. This is a perfect situation for an abstract superclass. You can take advantage of the similarities and declare all the graphic objects to inherit from the same abstract parent object (for example, GraphicObject) as shown in the following figure.



Classes Rectangle, Line, Bezier, and Circle Inherit from GraphicObject

First, you declare an abstract class, GraphicObject, to provide member variables and methods that are wholly shared by all subclasses, such as the current position and the moveTo method. GraphicObject also declares abstract methods for methods, such as draw or resize, that need to be implemented by all subclasses but must be implemented in different ways. The GraphicObject class can look something like this:

abstract class GraphicObject {

int x, y;

...

void moveTo(int newX, int newY) {

...

}

abstract void draw();

abstract void resize();

}

Each nonabstract subclass of GraphicObject, such as Circle and Rectangle, must provide implementations for the draw and resize methods:

class Circle extends GraphicObject {

void draw() {

...

}

void resize() {

...

}

}

class Rectangle extends GraphicObject {

void draw() {

...

}

void resize() {

...

}

}

## When an Abstract Class Implements an Interface

In the section on [Interfaces](https://docs.oracle.com/javase/tutorial/java/IandI/createinterface.html), it was noted that a class that implements an interface must implement *all* of the interface's methods. It is possible, however, to define a class that does not implement all of the interface's methods, provided that the class is declared to be abstract. For example,

abstract class X implements Y {

// implements all but one method of Y

}

class XX extends X {

// implements the remaining method in Y

}

In this case, class X must be abstract because it does not fully implement Y, but class XX does, in fact, implement Y.

## Class Members

An abstract class may have static fields and static methods. You can use these static members with a class reference (for example, AbstractClass.staticMethod()) as you would with any other class.

NOTES :

Binding variables and methods together is callled Encapsulation.

Parent class reference for child class objects -- Implicit casting

Abstract :

Abstarct methods : are methods with no boby

Concrete methods : are the methods with body

If you have any class which consists of methods without no body then we need to declare keyword "abstract" for the class ....which is callled as abstarct class.

Methods without body are also called abstract methods need to be declared with keyword "abstarct".

abstract class can consists of concrete methods and abstract methods.abstract methods are simply unimplemented methods.Implementationof the method body can be dealed by subclasses....

Objects are cant be created directly with the parent class...

Abstract classes can have the constructors, but it is called when we are creating the child class object.

and it allows to define all the access modifiers (public,private,protected and defualt ) for variables and methods.

When to use abstarct classess :

Whenever you want to provide common logic to be used by child logic and also we want to set some (abstract) methods to be implemented by child class just like setting rules for child class to implement.

Abstract and Interfaces are both act like Parent Classes, they will definitely have child classes where the methods are implemented and rules are given by parent.

INTERFACES :

Interface is 100% Abstract class.

Interface is a contract / setting the rules for Child class.

In Interface we cannot add constructor and we cannot reate object,Interface. hence we use static for calling the objects without creation.

None of the methods should contain body..that means all the methods have no body....that means all the methods are Abstract.

In interface no need to declare "Abstratc" keyword for methods and classes...default all the methods are abstract.

All the interface variables can be only public,static and final by default , we can declare the variablesin child class but if want to declare the variables in interface then we do need folloow the rules, that its called as Contract.

I

Static : to use variale without using the object...classname.variablename

final : not to re assigning ...making it as contacnt

In interface when we declare constant values no need to write (public static final) all defualt it takes as constant A=10;

Always constants it be declared in CAPS.

To provide implementation for the methods we create the child class instead of "Extends" we use "Implements" keyword

Private and protected access modifiers are not allowed for variables and methods too.

Interface contains only abstract methods.

When To USe :

Whenever we do not have any common lanuage but we just want to set rules for child classes we can use abstarct interface.

Also when multiple inheritance is needed in projects we can use Interface, because in java wecan not achieve multiple inheritance through Classes.

we need Interface when there is a situation, that we may need to add more methods to implement and you don’t want to distrub all existing child classes.

Java supports Multi level inheritance using class....but it only supports multiple Inheritance can only be done using Interfaces only.

class A extends B{

}

class B extends C{

}

Class C extends D{

}--- This means C extends B and A

D extends A,B and C

To avoid Deadly Death Diamond Problem java does not supports Multiple Inheritance.where as using Interface we can achieve multilevel Inheritance.

Multiple Inheritance :

Interface A -- test()

Interface B ---test()

Class C extentds A,B--- implementation is done in class both A and B has the same abstarct methods.

----------------------------------------------------

DATA TYPES :

Dattype :What kind of data variables to store

Java is statically types lang...we need to define what kind of data we are assigning.

dynamicall types means its decides at the execution time what kind of data it is.

there are 8 premitive data types in Java. Premitive means its just a keyword not the class or object in java.

other than these everything is Object type in Java.(part of class).

all premititive data types arein lowercase

All integer Decimal

byte - 8 bit -128 to 128 range float 32 bit char 'a' ascii charcters

short -16 bit double 64 bit boolean(T/F)

int - 32bit

long 64bit

String is not an datatype ..but is frequently used as data type,its a class in Java. it is set of charecters...or array of charectors.

if the values are not assigned for the Global variables then it assigns default values by java...where as for Local variable values need to be assigned.

If we are giving the data type again for the same varicable in the method then it take the local variable value.

LOOPS / CONDITIONAL /

--------------------------

http://javaconceptoftheday.com/static-binding-and-dynamic-binding-in-java/

http://javarevisited.blogspot.com/2011/12/final-variable-method-class-java.html

http://www.sitesbay.com/java/java-final-keyword

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.

Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

What is framework in java

* provides readymade architecture.
* represents set of classes and interface.
* is optional.

What is Collection framework

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

### Hierarchy of Collection Framework

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

hierarchy of 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. |

### Iterator interface

|  |
| --- |
| Iterator interface provides the facility of iterating the elements in forward direction only. |

#### Methods of Iterator interface

There are only three methods in the Iterator interface. They are:

1. **public boolean hasNext()** it returns true if iterator has more elements.
2. **public object next()** it returns the element and moves the cursor pointer to the next element.
3. **public void remove()** it removes the last elements returned by the iterator. It is rarely used.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Java ArrayList class Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.  The important points about Java ArrayList class are:   * Java ArrayList class can contain duplicate elements. * Java ArrayList class maintains insertion order. * Java ArrayList class is non synchronized. * Java ArrayList allows random access because array works at the index basis. * In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.   Java ArrayList class hierarchy Hierarchy of ArrayList class As shown in above diagram, Java ArrayList class extends AbstractList class which implements List interface. The List interface extends Collection and Iterable interfaces in hierarchical order. ArrayList class declaration Let's see the declaration for java.util.ArrayList class.   1. **public** **class** ArrayList<E> **extends** AbstractList<E> **implements** List<E>, RandomAccess, Cloneable, Serializable  Constructors of Java ArrayList  |  |  | | --- | --- | | **Constructor** | **Description** | | ArrayList() | It is used to build an empty array list. | | ArrayList(Collection 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. |  Methods of Java ArrayList  |  |  | | --- | --- | | **Method** | **Description** | | void add(int index, Object element) | It is used to insert the specified element at the specified position index in a list. | | boolean addAll(Collection c) | It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator. | | void clear() | It is used to remove all of the elements from this list. | | int lastIndexOf(Object o) | It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element. | | Object[] toArray() | It is used to return an array containing all of the elements in this list in the correct order. | | Object[] toArray(Object[] a) | It is used to return an array containing all of the elements in this list in the correct order. | | boolean add(Object o) | It is used to append the specified element to the end of a list. | | boolean addAll(int index, Collection c) | It is used to insert all of the elements in the specified collection into this list, starting at the specified position. | | Object clone() | It is used to return a shallow copy of an ArrayList. | | int indexOf(Object o) | It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element. | | void trimToSize() | It is used to trim the capacity of this ArrayList instance to be the list's current size. | |