Vectors

SMITA SANKHE

Assistant Professor

Department of Computer Engineering

smitasankhe@somaiya.edu

SOMAIYA VIDYAVIHAR UNIVERSITY

K J Somaiya College of Engineering

Somaiya TRUST

Arrays

- Array is a collection of similar type of elements that have contiguous memory location. **Java array** is an object the contains elements of similar data type. We can store only fixed set of elements in a java array.
- In java, array is an object.
- There are two types of array.
    - Single Dimensional Array
    - Multidimensional Array

Single dimensional Array

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

int a[]=new int[4];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;

//printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}}
int a[]={33,3,4,5};//declaration, instantiation and initialization done together
```

Multidimensional Array

- In Java, *multidimensional arrays* are actually arrays of arrays.
- **int**[][] arr=**new int**[3][3];//3 row and 3 columns

```
class Testarray3{
public static void main(String args[]){
 int arr[][]={{1,2,3},{2,4,5},{4,4,5}};   //declaring and initializing 2D array
for(int i=0;i<3;i++){  //printing 2D array
 for(int j=0;j<3;j++){
   System.out.print(arr[i][j]+" ");
 }
 System.out.println();
}

}}
```

Jagged Array

- It is a new feature supported by Java, where column size varies, ie each row may have varying length.

10 20 30

11 22 22 33 44

77 88

```
int twoD[][] = new int[3][];
twoD[0] = new int[3];
twoD[1] = new int[5];
twoD[2] = new int[2];
```

Array of Objects

```
class Student {
   int marks;
}
Student studentArray[] = new Student[7];
```

- The above statement creates the array which can hold references to seven Student objects. It doesn't create the Student objects themselves. They have to be created separately using the constructor of the Student class. The studentArray contains seven memory spaces in which the address of seven Student objects may be stored. If we try to access the Student objects even before creating them, run time errors would occur.

```
for ( int i=0; i<studentArray.length; i++) {
studentArray[i]=new Student();
}
```

- The above for loop creates seven Student objects and assigns their reference to the array elements. Now, a statement like the following would be valid.
- studentArray[0].marks=100;

Foreach loop

- JDK 1.5 introduced a new for loop known as foreach loop or enhanced for loop, which enables you to traverse the complete array sequentially without using an index variable.

Example:

```
public class TestArray
{
public static void main(String[] args)
{
 double[] myList = {1.9, 2.9, 3.4, 3.5};
// Print all the array elements
for (double element: myList)
{ System.out.println(element);
 }
 }
 }
```

Wrapper Classes

- Java is an object-oriented language and can view everything as an object. **Wrapper class in java** provides the mechanism *to convert primitive data types into objects and objects into primitive data types*. The automatic conversion of primitive into object is known and autoboxing and vice-versa unboxing. The object of the wrapper class contains or wraps its respective primitive data type**.**

*int k = 100; //*The **int** data type **k** is converted into an object, it1 using Integer class.

Autoboxing /wrapping

Integer it1 = new Integer(k);

int m = it1.intValue(); **Unboxing/Unwrapping**

*System.out.println(m*m); // prints 10000*

Wrapper Classes

- As pointed out earlier, collections cannot handle basic data types such as int, float. They can converted into object types by using the wrapper classes supported by java.lang package.

Basic Type	Wrapper Class
boolean	Boolean
char	Character
int	Integer
long	Long
float	Float
double	Double

Vectors

- Vector implements a **dynamic array of objects**.
- Vector proves to be very useful if you don't know the size of the array in advance or you just need one that can change sizes over the lifetime of a program.
- Vector can contain heterogeneous objects
- We **cannot store elements of primitive data type;** first it need to be converted to objects. A vector can store any objects.
- Its defined in ***java.util*** package.
- A vector has an initial capacity, if this capacity is reached then size of vector automatically increases.
- To traverse elements of a vector class we use **Enumeration** interface.
- Each vector tries to optimize storage management by maintaining a *capacity* and a *capacityIncrement* arguments.
- This class is a member of the Java Collections Framework.
- default initial capacity of vectors are 10.

Constructor and Description

1. **Vector( )** This constructor creates a default vector, which has an initial size of 10
2. **Vector(int size)**This constructor accepts an argument that equals to the required size, and creates a vector whose initial capacity is specified by size:
3. **Vector(int size, int incr)**This constructor creates a vector whose initial capacity is specified by size and whose increment is specified by incr. The increment specifies the number of elements to allocate each time that a vector is resized upward

Sample Program

```
/*This Java Example shows how to create an object of Java Vector. It also shows how to add elements to Vector
    and how get the same from Vector.*/
import java.util.Iterator;
import java.util.Vector;
 public class SimpleVectorExample
{
  public static void main(String[] args) {
    //create a Vector object
    Vector v = new Vector();
     v.add("1");
    v.add("2");
    v.add("3");   /*  Add elements to Vector using  boolean add(Object o) method    */
    System.out.println("Getting elements of Vector");
     System.out.println(v.get(0));
    System.out.println(v.get(1));
    System.out.println(v.get(2));    /*Use get method of Java Vector class to display elements of
                                      Vector.  */
  }
}
```

```
import java.util.*;
public class VectorDemo
{
 public static void main(String args[])
{
 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 Integer(2));
v.addElement(new Integer(3));
 v.addElement(new Integer(4));
System.out.println("Capacity after four additions: " + v.capacity());
```

```
v.addElement(new Double(5.45));
System.out.println("Current capacity: " + v.capacity());
 v.addElement(new Double(6.08));
System.out.println("Current capacity: " + v.capacity());
v.addElement(new Float(7.4));
System.out.println("Current capacity: " + v.capacity());
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() + " ");} }
```

Methods of Vector Class

1. boolean add(E e) This method appends the specified element to the end of this Vector.

2.void add(int index, E element) This method inserts the specified element at the specified position in this Vector.

3.boolean addAll(Collection<? extends E> c) This method appends all of the elements in the specified Collection to the end of this Vector.

4.boolean addAll(int index, Collection<? extends E> c) This method inserts all of the elements in the specified Collection into this Vector at the specified position.

5.void addElement(E obj) This method adds the specified component to the end of this vector, increasing its size by one.

6.int capacity() This method returns the current capacity of this vector.

7.void clear() This method removes all of the elements from this vector.

8.clone clone() This method returns a clone of this vector.

9.boolean contains(Object o) This method returns true if this vector contains the specified element.

10.boolean containsAll(Collection<?> c) This method returns true if this Vector contains all of the elements in the specified Collection.

11.void copyInto(Object[ ] anArray) This method copies the components of this vector into the specified array.

12. E elementAt(int index) This method returns the component at the specified index.

13.Enumeration<E> elements() This method returns an enumeration of the components of this vector.

14.void ensureCapacity(int minCapacity) This method increases the capacity of this vector, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument.

Ctd..

15.**boolean equals(Object o)** This method compares the specified Object with this Vector for equality.

16.**E firstElement()** This method returns the first component (the item at index 0) of this vector.

17.**E get(int index)** This method returns the element at the specified position in this Vector.

18**int hashCode()**This method returns the hash code value for this Vector.

19**int indexOf(Object o)**This method returns the index of the first occurrence of the specified element in this vector, or -1 if this vector does not contain the element.

20**int indexOf(Object o, int index)**This method returns the index of the first occurrence of the specified element in this vector, searching forwards from index, or returns -1 if the element is not found.

21**void insertElementAt(E obj, int index)**This method inserts the specified object as a component in this vector at the specified index.

22**boolean isEmpty()**This method tests if this vector has no components.

23**E lastElement()**This method returns the last component of the vector.

24**int lastIndexOf(Object o)**This method returns the index of the last occurrence of the specified element in this vector, or -1 if this vector does not contain the element.

25**int lastIndexOf(Object o, int index)** This method returns the index of the last occurrence of the specified element in this vector, searching backwards from index, or returns -1 if the element is not found.

26**E remove(int index)** This method removes the element at the specified position in this Vector.

27**boolean remove(Object o)** This method removes the first occurrence of the specified element in this Vector If the Vector does not contain the element, it is unchanged.

28**boolean removeAll(Collection<?> c)** This method removes from this Vector all of its elements that are contained in the specified Collection.

29**void removeAllElements()** This method removes all components from this vector and sets its size to zero.

30**boolean removeElement(Object obj)** This method removes the first occurrence of the argument from this vector.

31 **void removeElementAt(int index)** This method deletes the component at the specified index.

32 **protected void removeRange(int fromIndex, int toIndex)** This method removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.

33 **boolean retainAll(Collection<?> c)** This method retains only the elements in this Vector that are contained in the specified Collection.

34 **E set(int index, E element)** This method replaces the element at the specified position in this Vector with the specified element.

35 **void setElementAt(E obj, int index)** This method sets the component at the specified index of this vector to be the specified object.

36 **void setSize(int newSize)** This method sets the size of this vector.

37 **int size()** This method returns the number of components in this vector.

38 **List <E> subList(int fromIndex, int toIndex)** This method returns a view of the portion of this List between fromIndex, inclusive, and toIndex, exclusive.

39 **object[ ] toArray()**

This method returns an array containing all of the elements in this Vector in the correct order.

40 **<T> T[ ] toArray(T[ ] a)** This method returns an array containing all of the elements in this Vector in the correct order; the runtime type of the returned array is that of the specified array.

41 **String toString()** This method returns a string representation of this Vector, containing the String representation of each element.

42 **void trimToSize()** This method trims the capacity of this vector to be the vector's current size.

Differences between a Vector and an Array

- **Differences between a Vector and an Array**
  - A vector is a dynamic array, whose size can be increased, where as an array size can not be changed.
  - Reserve space can be given for vector, where as for arrays can not.
  - A vector is a class where as an array is not.
  - Vectors can store any type of objects, where as an array can store only homogeneous values.

  Advantages of Arrays:
  - Arrays supports efficient random access to the members.
  - It is easy to sort an array.
  - They are more appropriate for storing fixed number of elements

  Disadvantages of Arrays:
  - Elements can not be deleted
  - Dynamic creation of arrays is not possible
  - Multiple data types can not be stored

  Advantages of Vector:
  - Size of the vector can be changed
  - Multiple objects can be stored
  - Elements can be deleted from a vector

  Disadvantages of Vector:
  - A vector is an object, memory consumption is more.

- It is easy to sort an array.
- They are more appropriate for storing fixed number of elements
- **Disadvantages of Arrays:**
  - Elements can not be deleted
  - Dynamic creation of arrays is not possible
  - Multiple data types can not be stored

  Advantages of Vector:
  - Size of the vector can be changed
  - Multiple objects can be stored
  - Elements can be deleted from a vector

ArrayList

- ArrayList is a part of collection framework and is present in java.util package.
- It provides us with dynamic arrays in Java. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed.
- This class is found in java.util package.

```java
// Java program to demonstrate the
// working of ArrayList in Java

import java.io.*;
import java.util.*;

class ArrayListExample {
    public static void main(String[] args)
    {
        // Size of the
        // ArrayList
        int n = 5;

        // Declaring the ArrayList with
        // initial size n
        ArrayList<Integer> arrli
            = new ArrayList<Integer>(n);

        // Appending new elements at
        // the end of the list
        for (int i = 1; i <= n; i++)
            arrli.add(i);

        // Printing elements
        System.out.println(arrli);

        // Remove element at index 3
        arrli.remove(3);

        // Displaying the ArrayList
        // after deletion
        System.out.println(arrli);

        // Printing elements one by one
        for (int i = 0; i < arrli.size(); i++)
            System.out.print(arrli.get(i) + " ");
    }
}
```

[1, 2, 3, 4, 5]
[1, 2, 3, 5]
1 2 3 5

Object oriented concepts

- **Class -** A class can be defined as a template/blue print that describes the behaviors/states that object of its type support. A class is a group of objects that has common properties. class is the **logical entity** only. A class in java can contain:

1. **data member**
2. **method**
3. **constructor**
4. **block**
5. **class and interface**

Objects

- **Object -** Objects have states and behaviors. Object is the **physical** as well as **logical** entity. Objects are created using following keywords.

  By new keyword- New keyword is used to allocate memory at runtime.

  By newInstance() method

  By clone() method

  By factory method etc.

- An object that have no reference is known as annonymous object and it can be used when object is used only once.

Eg:- new Classname().

- **Object is an instance of a class.** An object has three characteristics:
  - **state:** represents data (value) of an object.
  - **behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.
  - **identity:** Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But,it is used internally by the JVM to identify each object uniquely.

Example

```
class Student1{
 int id;//data member (also instance variable)
 String name;//data member(also instance variable)
 public static void main(String args[]){
Student1 s1=new Student1();//creating an object of Stu
dent
  System.out.println(s1.id);
  System.out.println(s1.name);
 }
}
```

Ctd..

```
class Rectangle{
 int length;
 int width;

 void insert(int l,int w){
  length=l;
  width=w;
 }

 void calculateArea(){System.out.println(length*width);}

 public static void main(String args[]){
  Rectangle r1=new Rectangle(),r2=new Rectangle();//creating two objects

  r1.insert(11,5);
  r2.insert(3,15);

  r1.calculateArea();
  r2.calculateArea();
}
 }
```

Constructors

- A constructor **initializes an object** when it is created. It has the same name as its class and is syntactically similar to a method. However, constructors have no explicit return type.
- We use a constructor to give initial values to the instance variables defined by the class, or to perform any other startup procedures required to create a fully formed object.
- All classes have constructors, whether you define one or not, because Java automatically provides a default constructor that initializes all member variables to zero. However, once you define your own constructor, the default constructor is no longer used.

Types of Constructors

- There are two types of constructors:
  - **Default constructor** (no-arg constructor): Default constructor provides the default values to the object like 0, null etc. It will be invoked at the time of object creation.depending on the type. <class_name>(){}
  - **Parameterized constructor**: A constructor that have parameters. Parameterized constructor is used to provide different values to the distinct objects.
- There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.
  - By constructor
  - By assigning the values of one object into another
  - By clone() method of Object class

Method

- A Java method is a collection of statements that are grouped together to perform an operation. Syntax is

modifier returnType nameOfMethod (Parameter List) { // method body }

- Modifier is optional. There are two ways in which a method is called i.e. method returns a value or returning nothing

Si no	Constructor	Method
1	Constructor is used to initialize the **state** of an object.	Method is used to expose **behaviour** of an object.
2	Constructor must not have return type	Method must have return type.
3	Constructor is invoked implicitly.	Method is invoked explicitly.
4	The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in any case.
5	Constructor name must be same as the class name	Method name may or may not be same as class name.

parametarized constructor

•A default constructor does not have any parameter, but if you need, a constructor can have parameters. This helps you to assign initial value to an object at the time of its creation

```
class MyClass {
 int x; // Following is the constructor MyClass(int i ) {
 x = i;
 }
 }
public class ConsDemo {
 public static void main(String args[]) { MyClass t1 = new MyClass( 10 );
 MyClass t2 = new MyClass( 20 ); System.out.println(t1.x + " " + t2.x);
} }
```

This keyword

- **this** is a keyword in Java which is used as a reference to the object of the current class, with in an instance method or a constructor. Using *this* you can refer the members of a class such as constructors, variables and methods.

- usage of java this keyword**.**
    1. this keyword can be used to refer current class instance variable.
    2. this() can be used to invoke current class constructor.
    3. this keyword can be used to invoke current class method (implicitly)
    4. this can be passed as an argument in the method call.
    5. this can be passed as argument in the constructor call.
    6. this keyword can also be used to return the current class instance.

Destructor

- A **destructor** is a special member function of a class that is executed whenever an object of it's class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class.
- A destructor will have exact same name as the class prefixed with a tilde (~) and it can neither return a value nor can it take any parameters. Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc.

Finalize()

- The **finalize()** method is equivalent to a **destructor** of C++. When the job of an object is over, or to say, the object is no more used in the program, the object is known as **garbage**. The process of removing the object from a running program is known as **garbage collection**. finalize() method can be best utilized by the programmer to close the I/O streams, JDBC connections or socket handles etc.

smitasankhe@somaiya.edu

Access Specifiers in Java

- **Public:** A **class, method, constructor, interface** etc declared public can be accessed from any other class. Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe. However if the public class we are trying to access is in a different package, then the public class still need to be imported. Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.
- **Protected: Variables, methods and constructors** which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class. The protected access modifier **cannot be applied to class and interfaces**. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected. Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.
- **Private: Methods, Variables and Constructors** that are declared private can only be accessed within the declared class itself. Private access modifier is the most restrictive access level. **Class and interfaces cannot be private**. Variables that are declared private can be accessed outside the class if public getter methods are present in the class. Using the private modifier is the main way that an object encapsulates itself and hide data from the outside world.
- **Default:** Default access modifier means we do not explicitly declare an access modifier for a **class, field, method,** etc. A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.

Scope of access specifiers

Access Modifiers	Default	private	protected	public
Accessible inside the class	yes	yes	yes	yes
Accessible within the subclass inside the same package	yes	no	yes	yes
Accessible outside the package	no	no	no	yes
Accessible within the subclass outside the package	no	no	yes	yes

Final,finally and finalize

No	final	finally	finalize
1)	Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.	Finally is used to place important code, it will be executed whether exception is handled or not.	Finalize is used to perform clean up processing just before object is garbage collected.
2)	Final is a keyword.	Finally is a block.	Finalize is a method.