Object Oriented Programming in Java

Er.Sital Prasad Mandal

BCA- 2nd sem
Mechi Campus
Bhadrapur, Jhapa, Nepal
(Email : info.sitalmandal@gmail.com)
https://ctaljava.blogspot.com/

Text Book

1. Deitel & Dietel. -Java: How to-program-. 9th Edition. TearsorrEducation. 2011, ISBN: 9780273759168

2. Herbert Schildt. "Java: The CoriviaeReferi4.ic e 61 Seventh Edition. McGraw -Hill 2006, 1SBN; 0072263857

9. Understanding Core Packages

1. Using java.lang Package: java.lang.Math,
2. Wrapper classes and associated methods (Number, Double, Float; Integer, Byte; Short, Long; Character, Boolean);
3. Using java.util package: Core classes (
4. Vector,
5. Stack,
6. Dictionary,
7. Hashtable,
8. Enumerations,
9. Random Number Generation).

9. Understanding Core Packages

Package

A Package is a collection of similar Java entities such as classes, interfaces, sub-packages, exceptions, errors, and enums.

The same way Java classes and Interfaces are grouped under a Package.

As per Official Documentation, A *package* is a grouping of related types providing access protection and namespace management. Note that *types* refer to classes, interfaces, enumerations, and annotation types.

You can either use built-in packages or create your Package to group related classes and sub-packages together as a programmer.

Why use Packages

Using Packages in Java can be helpful in many ways:-

- Using packages in Java can aid in **Data Encapsulation** or Data Hiding.
- Using packages in Java creates a new **namespace** so the names of your Class and interfaces won't conflict with the type names in other packages.
- You can allow types within the Package to have **unrestricted access** to one another yet still restrict access for types outside the Package.

Using java.lang Package: java.lang.Math

Exploring in-built Java Packages

- So you now know how to create a user-defined package, how to compile files inside packages and the hierarchy of packages.
- Some packages are already defined in Java and are included in java software. These packages are called built-in packages.
- These packages come automatically with JDK/JRE download in the form of jar files.
- *You can view all the in-built packages by unzipping the rt.jar available in the lib folder of JRE, inside the Java directory.*

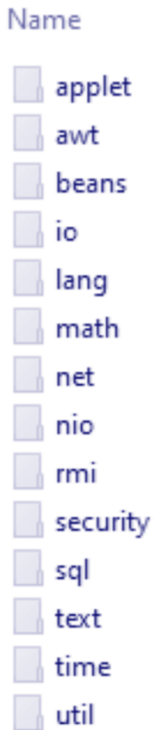
9. Understanding Core Packages

Using java.lang Package: java.lang.Math

How many packages are there in Java?

Broadly, there are two types of packages in Java, namely, built-in packages and user-defined packages.

There are 14 built-in Java packages. The in-built Java Packages list is shown below:-



Using java.lang Package: java.lang.Math

java.lang package

Among the in-built Java Packages, the java.lang package provides classes that are fundamental to the design of the Java programming language.

The most important Class in this Package is the Object class, which is the root of the class hierarchy and Class whose instances represent the classes at run time.

Furthermore, wrapper classes Boolean, Character, Integer, Float, and Double are also present in this Package.

This Package is automatically imported to each of the Java programs.

Some of the essential Java classes and interfaces present in the Java.lang package are listed below:-

Point to Remember

Using Static Methods

- Recall that the previous model of making method calls was this:
  - object.method(paramaters);
- With static methods, there is no object to call the method on. All info goes through the class to produce a result. And so, the model is this for static methods:
  - returnValue = ClassName.method(parameters);

Point to Remember

Temptations

- It might be tempting to do this...
  - `resultValue = someValue.sqrt();  // NOOOOOOO`
- This is wrong for two reasons...
  - The Math class methods cannot be called on other objects or variables.
    - The thing you want to operate on has to be passed in as a parameter.
  - Also, someValue is a variable, not an object, and it is not possible to call a method on a variable.

Using java.lang Package: java.lang.Math

java.lang package

Some of the essential Java classes and interfaces present in the Java.lang package are listed below:-

Class	Description
Object	Class Object is the root of the class hierarchy, every Class has Object as a superclass.
Thread	A thread is a thread of execution in a program. Thread class controls each thread in a multithreaded program.
Throwable	The Throwable class is the superclass of all errors and exceptions in Java language,
Math	The class Math contains methods for basic mathematical operations in Java

Using java.lang Package: java.lang.Math

The Math Class

- Java has many mathematical constants and functions available.
- There is a class built into Java called Math that contains all of these functions.
- This will only cover a part of the Math class. Complete information on Math can be found in the Java API documentation.
- Just add this line at the top of your program:
  - import java.lang.Math;

9. Understanding Core Packages

Using java.lang Package: java.lang.Math

Math.max(*x*,*y*)

The `Math.max(x,y)` method can be used to find the highest value of *x* and *y*:

Example: `Math.max(5, 10);`

Math.min(*x*,*y*)

The `Math.min(x,y)` method can be used to find the lowest value of *x* and *y*:

Example: `Math.min(5, 10);`

Math.sqrt(*x*)

The `Math.sqrt(x)` method returns the square root of *x*:

Example: `Math.sqrt(64);`

Math.abs(*x*)

The `Math.abs(x)` method returns the absolute (positive) value of *x*:

Example: `Math.abs(-4.7);`

Random Numbers

`Math.random()` returns a random number between 0.0 (inclusive), and 1.0 (exclusive):

Example: `Math.random();`

To get more control over the random number, for example, if you only want a random number between 0 and 100, you can use the following formula:

```
int randomNum = (int)(Math.random() * 101); // 0 to 100
```

Math.pow(x,y)

Returns of value of the first argument raised to the power of the second argument.

```
System.out.println(Math.pow(2, 4)); //16
```

Using java.lang Package: java.lang.Math

Schematics!

- double exp(double x)
  - This returns e raised to the x power
- double pow(double x, double y)
  - This returns x raised to the y power
- double log(double x)
  - This returns the natural logarithm of x, ln(x)
- double sqrt(double x)
  - This returns the square root of x.
- double abs(double x)
  - This returns the absolute value of x, |x|

- toDegrees() : Converts the given angle in radians to degrees.
- toRadians() : Reverse of toDegrees.
- Trigonometric Functions
  - sin(a)
  - cos(a)
  - tan(a)
    - **All trigonometric functions are computed in radians.**
- Arc Trigonometric Functions
  - asin(a)
  - acos(a)
  - atan(a)

9. Understanding Core Packages

Using java.lang Package: java.lang.Math

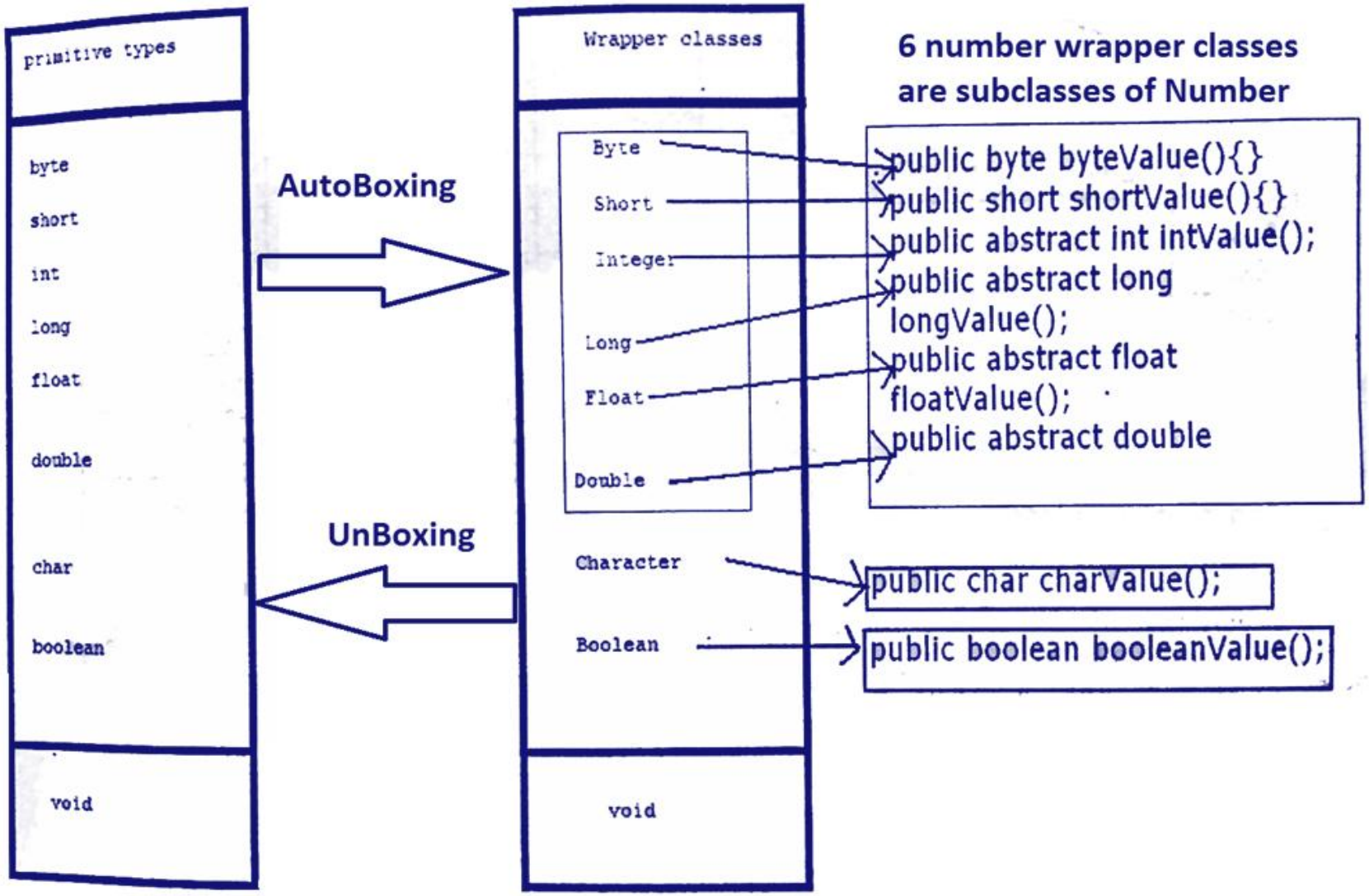
```
public class MathDemo {
    public static void main ( String[] args ) {
        System.out.println("Exploring Java packages list");
        int a = 2, b = 3;  // Using methods defined inside Math class of Java
        int sum = Math.addExact(a, b);
        int multiply = Math.multiplyExact(a, b);
        System.out.println("Sum is: " + sum );
        System.out.println("Multiplication is: " + multiply);

        // Using methods defined inside Integer Wrapper class of Java
        int c = 2;  // Use to convert an integer to string
        System.out.println("Using toString(c): " + c);

        // Use to convert an integer to Binary String
   System.out.println("Using toBinaryString(c): " + Integer.toBinaryString(c));
   }
}
```

Wrapper classes

- Each of Java's eight primitive data types has a class dedicated to it.
- These are known as *wrapper classes*, because they "wrap" the primitive data type into an object of that class.
- there is an Integer class that holds an int variable.

9. Understanding Core Packages

Assignment Wrapper classes & associated methods



9. Understanding Core Packages

Wrapper classes

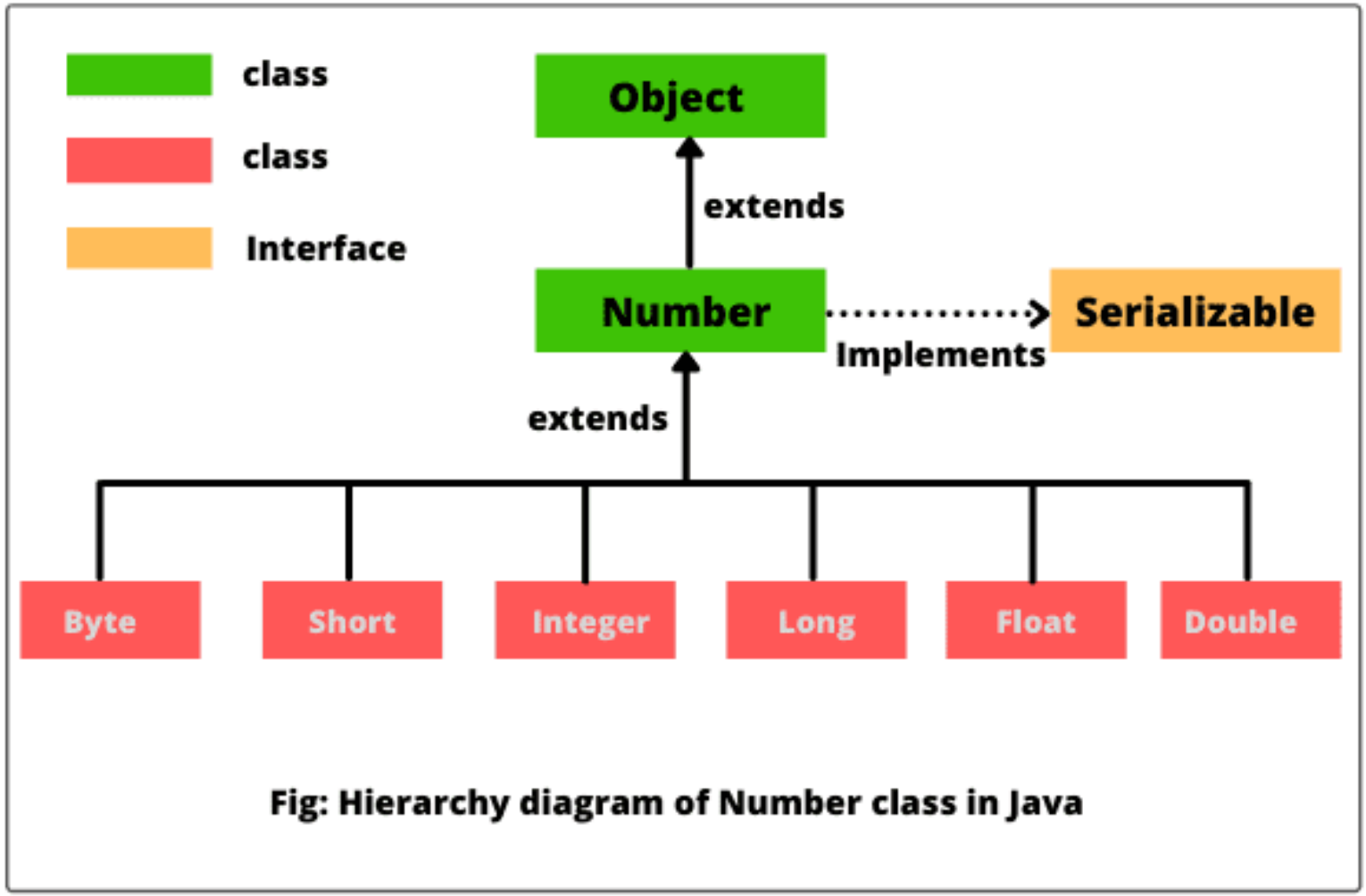


Fig: Hierarchy diagram of Number class in Java

Wrapper classes

- Wrapper Classes have Objects Defined as Follows :
  - Integer
    - Integer x = new Integer(value);
  - Long
    - Long x = new Long(value);
  - Double
    - Double x = new Double (value);
  - Float
    - Float x = new Float(value);
  - Char
    - Character x = new Character(value);

Wrapper classes & associated methods

- Wrapper Classes have a lot of methods in common :
  - toString() Method :
    - For Example :
      - String s = Integer.toString(5);
      - String s = Character.toString('a');

```
String  s = " 10.6f ";

float x = Float . parseFloat (s);
```

data type — wrapper class — method name

```
System.out.println(x);  // 10.6
```

  - parse Method : Converts String to an Int , float, double ,..
    - Int x = Integer.parseInt("1234");
    - double x = Double.parseDouble("12.1545");
  - Minimum and Maximum Values of a Primitive type
    - Int min = Integer.MIN_VALUE; //min =-2147483648
    - Int max = Integer.MAX_VALUE; // max = 2147483647
    - float maxv = Float.MAX_VALUE; //maxv = 3.4028235E38

Wrapper classes & associated methods

- Converting between primitive data types :
  - doubleValue() returns the value of this type as an double.
  - floatValue() returns the value of this type as a float.
  - intValue() returns the value of this type as an int.
  - longValue() returns the value of this type as a long.
    - For Example
      - int x = 15;
      - float y = x.floatValue();

Wrapper classes & associated methods

- Converting to another number system :
  - toBinaryString(a) : Converts a into binary string.
  - toHexString(a) : Converts a into hexadecimal string.
  - toOctalString(a) : Converts a into octal String.
  - For Example :
    - String s = Integer.toBinaryString(10);

Wrapper classes & associated methods

Advantages of Java Wrapper Classes

1. Java Wrapper Classes convert primitive data type into Object data type.
2. Java Wrapper Classes are very helpful in Java Collection Framework.
   Because most of the Java collection classes accepts object data type.
3. Java Wrapper Classes needed to use the classes of java.util package.

Wrapper classes

What is the Autoboxing in Java?

Automatic conversion of primitive data type to the object data type with corresponding Java wrapper class is called as Autoboxing.

For example convert primitive data type int to Java wrapper class Integer.

```
public class AutoboxingExample
{
    public static void main(String args[])
    {
        int intval = 100;
        // AUTOBOXING
        // AUTOMATIC CONVERSION FROM PRIMITIVE DATA TYPE TO WRAPPER CLASS OBJECT
        Integer intobj = intval;
        System.out.println("Autoboxing Value : "+intobj);
    }
}
```

Wrapper classes

What is the Unboxing in Java?

Automatic conversion of Java wrapper class object to the primitive data type is called as Unboxing.

For example convert Java wrapper class Integer to primitive data type int.

Java Unboxing example

```
public class UnboxingExample
{
    public static void main(String args[])
    {
        Integer intobj = new Integer(100);
        // UNBOXING
       // AUTOMATIC CONVERSION FROM WRAPPER CLASS OBJECT TO PRIMITIVE DATA TYPE
        int intval = intobj;
        System.out.println("Unboxing Value : "+intval);
    }
}
```

Using java.util package

java.util Package

The java.util package contains the collection framework, legacy collection classes, event model, date and time facilities, and miscellaneous utility classes like string tokenizer, a random-number generator, and a bit array.

Java directly depends on several classes in this package

1. The **HashTable** class for implementing hashtables
2. **Enumeration** interface for iterating through a collection of elements
3. **StringTokenizer** class for parsing strings into distinct tokens
4. The **Collection Framework** contains inbuilt implementations of various data structures and algorithms.
5. It can be use for base64 encoding and decoding.

Using java.util package

Core classes

1. Vector
2. Stack
3. Dictionary
4. Hashtable
5. Enumerations
6. Random Number Generation

Using java.util package

Vector

Size of an array is fixed for the duration of the execution program.
Vector is a more flexible data structure which allows its size to be changed.

- A vector class is part of the java.util package
- It can dynamically shrink or grow as needed
- A data element can be inserted into or removed from any location of a vector with a single method invocation.
- The elements of a vector must be objects (cannot be primitive types - **if you need to store a primitive type, you need to use appropriate wrapper classes**).
- It provides several instance methods to manage the list of objects
- A Vector is basically the same as an ArrayList, but Vector methods are synchronized for thread safety.

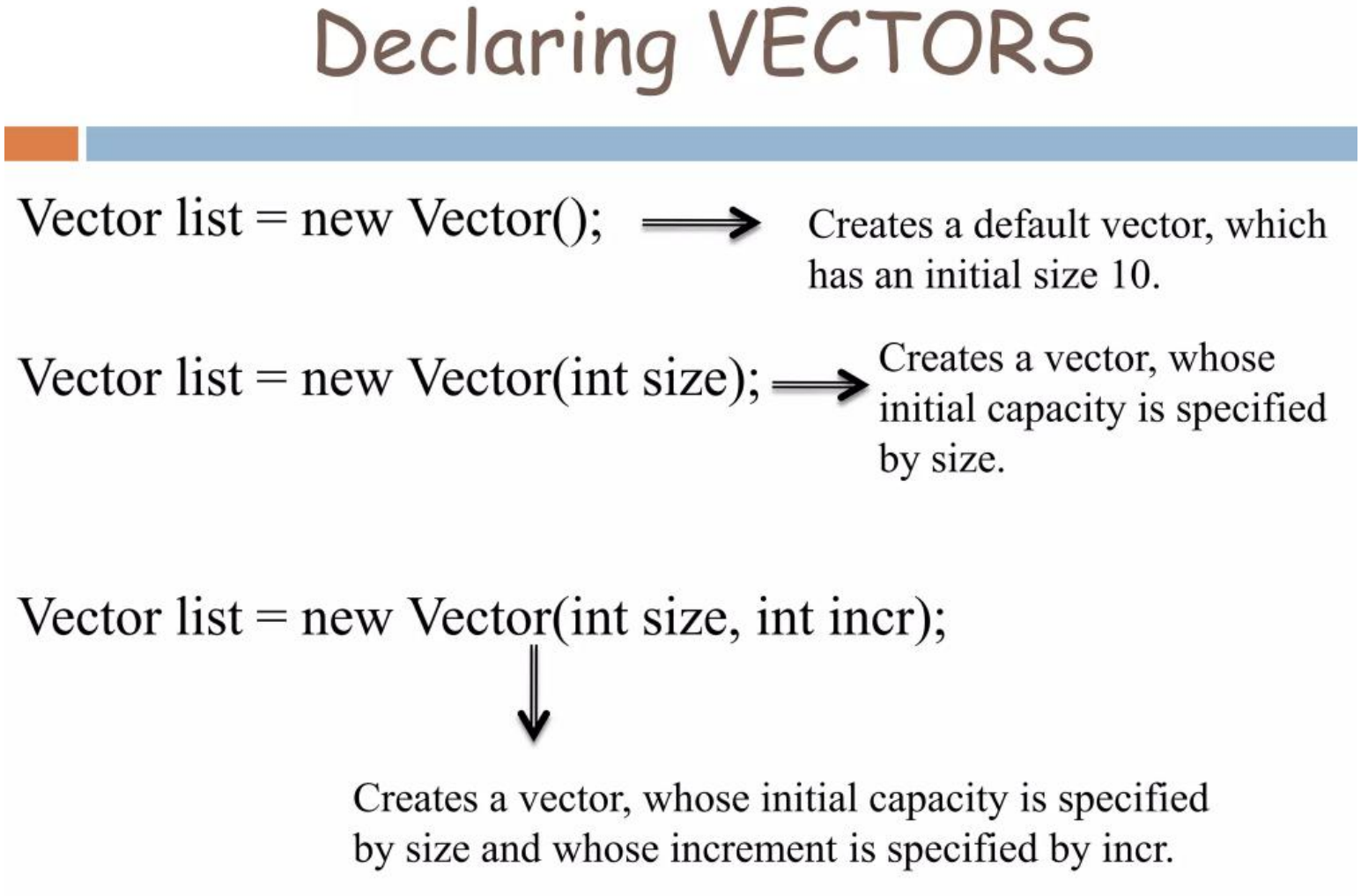
Using java.util package

Vector

- The Vector implements List interface.
- The Vector is a legacy method (1. Enumeration. 2. Iterator).
- The Vector is synchronized.
- it is rarly used in the non-thread application.
- It also lead to poor performance.
- Vector is similar to an ArrayList.
- Like ArrayList, Vector also maintains the insertion order.

Using java.util package

Constructor Vector



9. Understanding Core Packages

Using java.util package

Vector Methods

void **addElement**(Object *element*)	The object specified by *element* is added to the vector
int **capacity()**	Returns the capacity of the vector
boolean **contains**(Object *element*)	Returns **true** if element is contained by the vector, else **false**
void **copyInto**(Object *array[]*)	The elements contained in the invoking vector are copied into the array specified by array[]
elementAt(int *index*)	Returns the element at the location specified by *index*
Object **firstElement().**	Returns the first element in the vector

void **insertElementAt**(Object *element*, int *index*)	*Adds element* to the vector at the location specified by *index*
boolean **isEmpty()**	Returns **true** if Vector is empty, **else false**
Object **lastElement()**	Returns the last element in the vector
void **removeAllElements()**	Empties the vector. After this method executes, the size of vector is zero.
void **removeElementAt**(int *index*)	Removes element at the location specified by *index*
void **setElementAt**(Object *element*, int *index*)	The location specified by *index* is assigned *element*

void **setSize**(int *size*)	*Sets the number of elements in the vector to size. If the new size is less than the old size, elements are lost. If the new size is larger than the old, null elements are added*
int **size()**	Returns the number of elements currently in the vector

Using java.util package

Important Point to Remember of Vector

- There are methods to add , retrieve, insert in the middle of the vector, or remove elements

Vector
+ Vector() + Vector(in size : int) + addElement(in o : Object) + elementAt(in index : int) : Object + insertElementAt(in o : Object, in x : int) + indexOf(in o : Object) : int + lastIndexOf(in o : Object) : int + removeElementAt(in index : int) + size( ) : int

Using java.util package

Vectors - API documentation (partial listing)

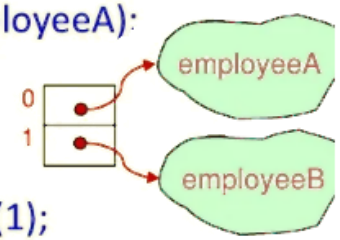
```
public class Vector extends Object implements Cloneable, Serializable
{
  // constructors
  public Vector (int initialCapacity);
  public Vector();

  // Instance methods for different functions
  public final void addElement (Object obj); - insert element at next free location
  public final int  capacity();   - return capacity of vector
  public final Object elementAt(int index);- return object stored at the position 'index'
  public final Object firstElement(); - return first element (object at vector position 0)
  public final int  indexOf(Object elem); - return index of the given element in vector
  public final Object lastElement(); - return last element (object at position size-1)
  public final synchronized boolean removeElement(Object obj); - removes specified object
  public final synchronized void removeAllelements(); - removes all objects
  public final int size(); - return number of objects stored in the vector
  public final void trimToSize(); - reduce capacity of the vector to the number of elements
}
```

Allows specification of initial capacity of vector

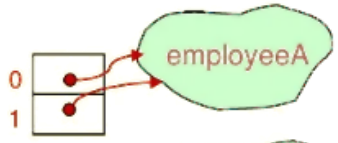
Example 1 of Vector: Exercise 1, Question

Use the Vector class to store a collection of employee:

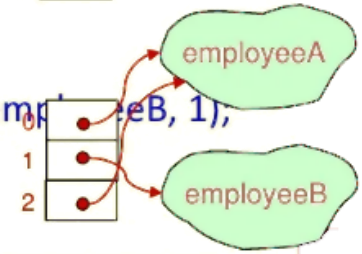
- import stmt: `import java.util.*;`
- data declaration: `Vector employees = new Vector();`
- add existing objects:
```
employees.addElement(employeeA);
employees.addElement(employeeB);
```



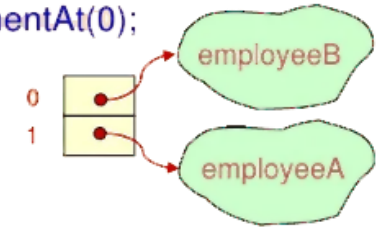
- access 2nd element:
```
anEmployee = (Employee) employees.elementAt(1);
```
- change an element:
```
employees.setElementAt(employeeA, 1);
```



- insert an element: `employees.insertElementAt(em...eB, 1);`



- remove an element: `employees.removeElementAt(0);`



Example 2 of Vector: Exercise 2, Question

- Use vector to store a collection of data that contains texts (i.e. faculty names).
- Import statement:

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

- Declare a `Vector` object of faculty:

```
Vector faculty = new Vector();
```

  - A `Vector` of size 0 is created.
- Add new element to a data collection:

```
faculty.addElement ("FSKSM");
faculty.addElement ("FS");
faculty.addElement ("FKE");
faculty.addElement ("FAB");
```

  - The `Vector` will contain three elements in location 0, 1, and 2.
- Display the content of vector:

```
System.out.println(faculty);
```

- Display the size of the vector:

```
System.out.println(faculty.size());
```

- To insert new element at the third element of the vector:

```
faculty.insertElementAt ("FPPSM", 2);
```

  - `"FPPSM"` is stored in between `"FS"`, and `"FKE"`.
- Display the content of vector:

```
System.out.println(faculty);
```

- Display the size of the vector:

```
System.out.println(faculty.size());
```

- To remove element with content `"FKE"` in the vector:

```
faculty.removeElement ("FKE");
```

- To remove element at the third element of the vector:

```
faculty.removeElementAt (1);
```

  - `"FS"` will be deleted.
- To display the content of vector at the specified element:

```
System.out.println (faculty.elementAt(1));
```

  - `"FPPSM"` will be displayed.
- To change the value of specific element of a vector

```
faculty.setElementAt("FKSG",2);
```

  - The content of element three will be replaced by `"FKSG"`.

Example 3 of Vector: Lab Exercise

```
import java.util.*;

public class VectorClassExample {

    public static void main ( String[] args ) {

        Vector list = new Vector();

        list.add(10);
        list.add(30);
        list.add(0, 100);
        list.addElement(50);
        list.insertElementAt(10, 4);

        System.out.println("Vector => " + list);

        System.out.println("get(2) => " + list.get(2));

        System.out.println("firstElement() => " + list.firstElement());

        System.out.println("indexOf(50) => " + list.indexOf(50));

        System.out.println("contains(30) => " + list.contains(30));

        System.out.println("capacity() => " + list.capacity()); //Default 10

        System.out.println("size() => " + list.size());

        System.out.println("isEmpty() => " + list.isEmpty());

    }
}
```

```
Vector => [100, 10, 30, 50, 10]
get(2) => 30
firstElement() => 100
indexOf(50) => 3
contains(30) => true
capacity() => 10
size() => 5
isEmpty() => false
```

Vector

PROGRAMMING EXAMPLE

- Write a program that accepts a shopping list of five items from the command line and stores them in a vector.
- Modify the program to accomplish the following:-

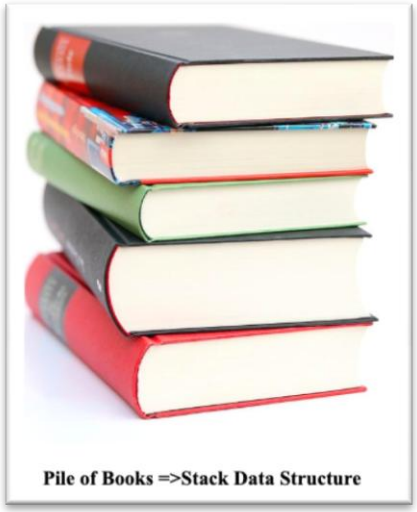
1. To delete an item in the list.
2. To add an item at a specified position in the list.
3. To add an item at the end of the list.
4. To print the contents of the vector.

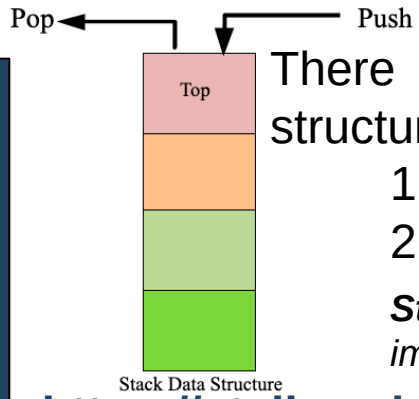
9. Understanding Core Packages

Using java.util package

Stack

- Stack class in Java is a special kind of data structure which can be thought a linear structure represented by physical stack or pile.
- In this data structure, we can add or remove an element from **the top of the structure**.
- Java Stack is the legacy Collection class. It extends the Vector class to support the **LIFO**(Last In First Out) operations.
- Stack class is also the List implementation class but does NOT support all the operations of Vector or List.
- As Stack supports LIFO, it is also known as **LIFO Lists**.

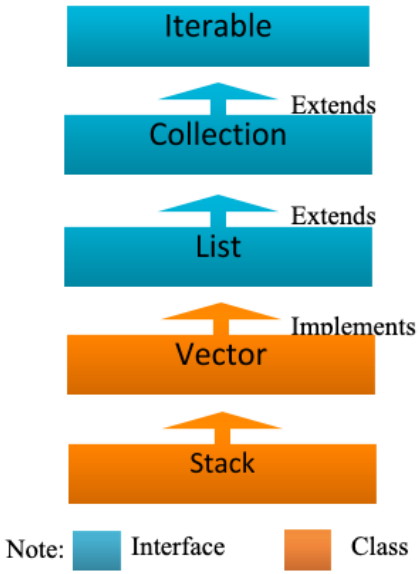




There are two types of operations in Stack data structure, i.e.,

1. push (inserting element into the stack)
2. pop (removing items from the stack).

***Stack** class is a subclass of **Vector,** and this **Vector** class implements List, **Collection,** and **Iterable** interfaces.*



Using java.util package

Stack Method

The Stack class in java has the following constructor.

S. No.	Constructor with Description
1	**Stack( )** It creates an empty Stack.

The Stack class in java has the following methods.

S.No.	Methods with Description
1	**Object push(Object element)** It pushes the element onto the stack and returns the same.
2	**Object pop( )** It returns the element on the top of the stack and removes the same.
3	**int search(Object element)** If element found, it returns offset from the top. Otherwise, -1 is returned.
4	**Object peek( )** It returns the element on the top of the stack.
5	**boolean empty()** It returns true if the stack is empty, otherwise returns false.

Stack Example

```
import java.util.Stack;
public class StackDemo {
    public static void main(String a[]){
        Stack stack = new Stack();
        System.out.println("Empty stack : "  + stack);
        System.out.println("Empty stack : "  + stack.isEmpty());
        stack.push(100);
        stack.push(102);
        stack.push(103);
        stack.push(104);
        System.out.println("Stack elements are : "  + stack);
        System.out.println("Stack: Top Element : "  + stack.peek());
        System.out.println("Stack: Pop Operation : "  + stack.pop());
        System.out.println("Stack: After Pop Operation : "  + stack);
        System.out.println("Stack : search() : "  + stack.search(100));
        System.out.println("If Stack is empty : "  + stack.isEmpty());
    }
}
```

```
Empty stack : []
Empty stack : true
Stack elements are : [100, 102, 103, 104]
Stack: Top Element : 104
Stack: Pop Operation : 104
Stack: After Pop Operation : [100, 102, 103]
Stack : search() Operation : 3
If Stack is empty : false
```

Using java.util package

Dictionary

- In java, the package **java.util** contains a class called **Dictionary** which works like a Map. The Dictionary is an abstract class used to store and manage elements in the form of a pair of key and value.

- The Dictionary stores data as a pair of key and value. In the dictionary, each key associates with a value. We can use the key to retrieve the value back when needed.

  - ❑ The Dictionary class is no longer in use, it is obsolete.
  - ❑ As Dictionary is an abstract class we can not create its object. It needs a child class like Hashtable.

Constructors

The following is the only constructor of the Dictionary class.

Dictionary() : Sole constructor.

Using java.util package

Dictionary

S. No.	Methods with Description
1	**Dictionary( )** It's a constructor.
2	**Object put(Object key, Object value)** Inserts a key and its value into the dictionary. Returns null on success; returns the previous value associated with the key if the key is already exist.
3	**Object remove(Object key)** It returns the value associated with given key and removes the same; Returns null if the key does not exist.
4	**Object get(Object key)** It returns the value associated with given key; Returns null if the key does not exist.
5	**Enumeration keys( )** Returns an enumeration of the keys contained in the dictionary.
6	**Enumeration elements( )** Returns an enumeration of the values contained in the dictionary.
7	**boolean isEmpty( )** It returns true if dictionary has no elements; otherwise returns false.
8	**int size( )** It returns the total number of elements in the dictionary.

9. Understanding Core Packages

Dictionary

```
import java.util.*;
public class DictionaryExample {
    public static void main(String args[]) {
        Dictionary dict = new Hashtable();
        dict.put(1, "Ram");
        dict.put(2, "Sita");
        dict.put(3, "Roshan");
        dict.put(4, "Vijaya");
        dict.put(5, "Ramesh");
        System.out.println("Dictionary\n=> " + dict);
        // keys()
        System.out.print("\nKeys in Dictionary\n=> " + dict.keys());
        for (Enumeration i = dict.keys(); i.hasMoreElements();)
        {
            System.out.print(" " + i.nextElement());
        }
        // Return the eneumeration of dictionary using elements() to get value
        for (Enumeration e = dict.elements(); e.hasMoreElements();)
        {
            System.out.println("\n" + e.nextElement());
        }
        //get()
        System.out.println("\n\nValue associated with key 3 => " + dict.get(3));
        System.out.println("Value associated with key 30 => " + dict.get(30));
        // remove the element 99 using remove() method
        System.out.println(" Remove the element : " + dict.remove("Roshan"));
        //size()
        System.out.println("\nDictionary has size: " + dict.size());
        //isEmpty()
        System.out.println("\nIs Dictionary empty?  " + dict.isEmpty());
    }
}
```

Dictionary
=> {5=Ramesh, 4=Vijaya, 3=Roshan, 2=Sita, 1=Ram}
Keys in Dictionary
=> java.util.Hashtable$Enumerator@7530d0a 5 4 3 2 1
Ramesh
Vijaya
Roshan
Sita
Ram
Value associated with key 3 => Roshan
Value associated with key 30 => null
Remove the element : null
Dictionary has size: 5
Is Dictionary empty? False

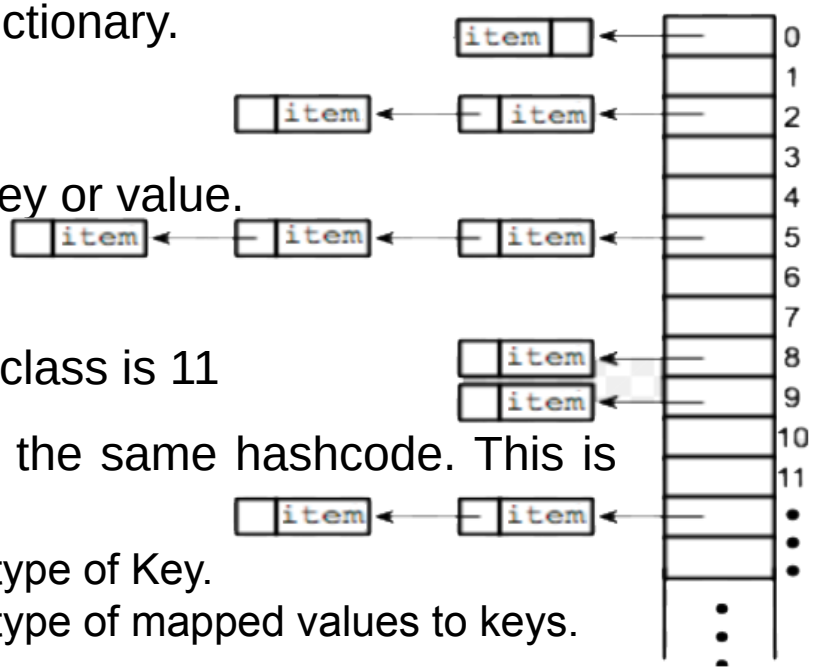
9. Understanding Core Packages

Using java.util package

Hashtable

Its connects a variable's name to its memory location.

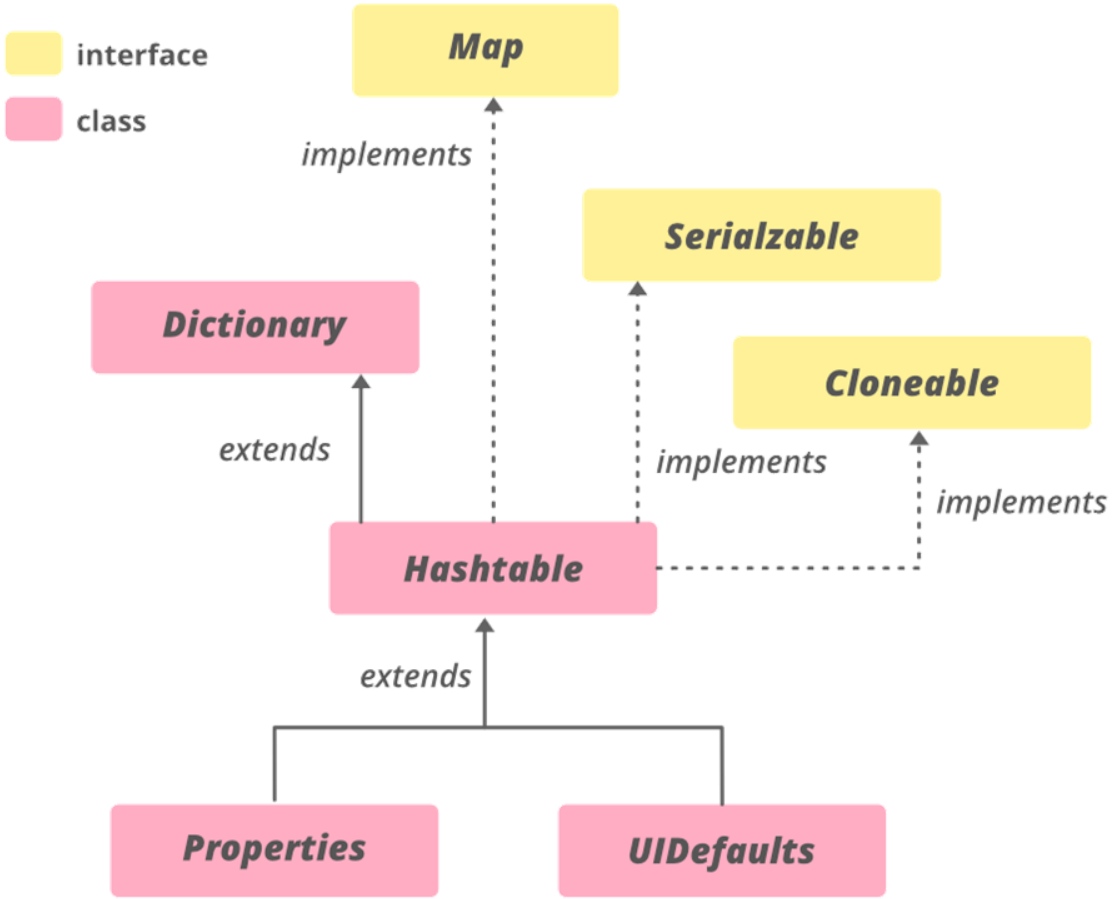
- A Hashtable is the array of a list. Each list is known as the bucket. The position of the bucket is identified by calling a hashcode() method. The Hashtable contains values based on the key.
- Hashtable stores key/value pair in the hash table.
- The Hashtable class is a concrete class of Dictionary.
- Java Hashtable class contains unique items.
- Java Hashtable class doesn't allow the null key or value.
- Java Hashtable class is synchronized.
- The initial by default capacity of a Hashtable class is 11
- It is possible that two unequal Objects have the same hashcode. This is called a **collision**..



Here,
'K' is a type of Key.
'V' is a type of mapped values to keys.

Using java.util package

Hashtable



Using java.util package

Hashtable

S. No.	**Constructor** with Description
1	**Hashtable( )** It creates an empty hashtable with the default initial capacity 11.
2	**Hashtable(int capacity)** It creates an empty hashtable with the specified initial capacity.
3	**Hashtable(int capacity, float loadFactor)** It creates an empty hashtable with the specified initial capacity and loading factor(0.75).
4	**Hashtable(Map m)** It creates a hashtable containing elements of Map m.

Using java.util package

Hashtable

S. No.	Methods with Description
1	**put(K key, V value)** It inserts the specified key and value into the hash table.
2	**get(Object key)** It returns the value associated with the given key.
3	**int hashCode()** It returns the hash code of the hashtable.
4	**Object clone()** It returns a shallow copy of the Hashtable.
5	**replace(K key, V value)** It replaces the specified value for a specified key.
6	**boolean isEmpty( )** It returns true if Hashtable has no elements; otherwise returns false.
7	**int size( )** It returns the total number of elements in the Hashtable.
8	**void clear()** It is used to remove all the elements of a Hashtable.

Hashtable

```
import java.util.*;
public class HashtableExample {
    public static void main(String[] args) {

        Hashtable table = new Hashtable();
        //add keys and values to the table
        //put(key, value)
        table.put(1,"one");
        table.put(2,"two");
        table.put(3,"three");
        table.put(4,"four");
        table.put(5,"five");

        System.out.println("Hashtable => " + table);
        //get(key)
        System.out.println("\nValue associated with key 3 => " + table.get(3));
        //keySet()
        System.out.println("\nKeys => " + table.keySet());
        //values()
        System.out.println("\nValues => " + table.values());
        //entrySet()
        System.out.println("\nKey, Value pairs as a set => " + table.entrySet());
        //hashCode()
        System.out.println("\nHash code => " + table.hashCode());
        //size()
        System.out.println("\nTotal number of elements => " + table.size());
        //isEmpty()
        System.out.println("\nEmpty status of Hashtable => " + table.isEmpty());
    }
}
```

```
Hashtable => {5=five, 4=four, 3=three, 2=two, 1=one}
Value associated with key 3 => three
Keys => [5, 4, 3, 2, 1]
Values => [five, four, three, two, one]
Key, Value pairs as a set => [5=five, 4=four, 3=three, 2=two, 1=one]
Hash code => 116857387
Total number of elements => 5
Empty status of Hashtable => false
```

Using java.util package

Enumerations

In java, an **Enumeration** is a list of named constants. The enum concept was introduced in Java SE 5 version. The enum in Java programming the concept of enumeration is greatly expanded with lot more new features compared to the other languages like C, and C++.

In java, the enumeration concept was defined based on the class concept. When we create an enum in java, it converts into a class type. This concept enables the java enum to have constructors, methods, and instance variables.

In java, an enum can be defined outside a class, inside a class, but not inside a method.

All the constants of an enum are ***public***, ***static***, and ***final***. As they are static, we can access directly using enum name.

The main objective of enum is to define our own data types in Java, and they are said to be enumeration data types.

Using java.util package

Enumerations

Creating enum in Java

```
enum WeekDay {
    MONDAY, TUESDAY, WEDNESSDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY
}

public class EnumerationExample {

    public static void main ( String[] args ) {
        WeekDay day = WeekDay.FRIDAY;
        System.out.println("Today is " + day);
        System.out.println("\nAll WeekDays: ");
        for (WeekDay d : WeekDay.values())
            System.out.println(d);
    }
}
```

- Every enum is converted to a class that extends the built-in class **Enum**.
- Every constant of an enum is defined as an object.
- As an enum represents a class, it can have methods, constructors. It also gets a few extra methods from the Enum class, and one of them is the **values()** method.

9. Understanding Core Packages

Using java.util package

Enumerations Constructors

EnumExample.java | EnumerationExample.java

```
1
2  enum WeekDay{
3      MONDAY, TUESDAY, WEDNESSDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY("Holiday");
4
5      String msg;
6
7      WeekDay(){
8          System.out.println("Default constructor!");
9      }
10
11     WeekDay(String str){
12         System.out.println("Parameterized constructor!");
13         msg = str;
14     }
15 }
16
17
18 public class EnumerationExample {
19
20     public static void main(String[] args) {
21
22         WeekDay day = WeekDay.SUNDAY;
23
24         System.out.println("\nToday is " + day + " and its " + day.msg);
25
26     }
27
28 }
```

```
Default constructor!
Default constructor!
Default constructor!
Default constructor!
Default constructor!
Default constructor!
Parameterized constructor!

Today is SUNDAY and its Holiday
```

Using java.util package

Methods in Java enum

EnumExample.java | *EnumerationExample.java

```
1
2  enum WeekDay{
3      MONDAY, TUESDAY, WEDNESSDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY("Holiday");
4
5      String msg;
6
7      WeekDay(){
8          System.out.println("Default constructor!");
9      }
10
11     WeekDay(String str){
12         System.out.println("Parameterized constructor!");
13         msg = str;
14     }
15
16     void printMessage() {
17         System.out.println("Today is also " + msg);
18     }
19 }
20
21
22 public class EnumerationExample {
23
24     public static void main(String[] args) {
25
26         WeekDay day = WeekDay.SUNDAY;
27
28         System.out.println("\nToday is " + day);
29         day.printMessage();
30     }
31 }
32
```

```
Default constructor!
Default constructor!
Default constructor!
Default constructor!
Default constructor!
Default constructor!
Parameterized constructor!

Today is SUNDAY
Today is also Holiday
```

Using java.util package

Random Number Generation

The **Random** is a built-in class in java used to generate a stream of pseudo-random numbers in java programming.
The Random class is available inside the **java.util** package.

The Random class implements **Serializable**, **Cloneable** and **Comparable** interface.

- The **Random** class is a part of java.util package.
- The **Random** class provides several methods to generate random numbers of type integer, double, long, float etc.
- The **Random** class is thread-safe.
- Random number generation algorithm works on the seed value. If not provided, seed value is created from system nano time.

Random Numbers are commonly used in creating applications like Dice for a board game, Gambling Program, etc.

In Java, Random Numbers can be generated using 3 ways:

1. Math.random method
2. java.util.Random class
3. ThreadLocalRandom class

Using java.util package

Random Number Generation

S.No.	Constructor with Description
1	**Random()** It creates a new random number generator.
2	**Random(long seedValue)** It creates a new random number generator using a single long seedValue.

Using java.util package

Random Number Generation

S.No.	Methods with Description
1	**int next(int bits)** It generates the next pseudo-random number.
2	**Boolean nextBoolean()** It generates the next uniformly distributed pseudo-random boolean value.
3	**double nextDouble()** It generates the next pseudo-random double number between 0.0 and 1.0.
4	**void nextBytes(byte[] bytes)** It places the generated random bytes into an user-supplied byte array.
5	**float nextFloat()** It generates the next pseudo-random float number between 0.0 and 1.0..
6	**int nextInt()** It generates the next pseudo-random int number.
7	**int nextInt(int n)** It generates the next pseudo-random integer value between zero and n.
8	**long nextLong()** It generates the next pseudo-random, uniformly distributed long value.

Using java.util package

Random Number Generation

```
import java.util.Random;

public class RandomClassExample {
    public static void main(String[] args) {
        Random rand = new Random();
        System.out.println("Integer random number: " + rand.nextInt());
        System.out.println("Integer random number from 0 to 50 is: " +
rand.nextInt(50));
        System.out.println("Boolean random value: " + rand.nextBoolean());
        System.out.println("Double random number: " + rand.nextDouble());
        System.out.println("Float random number: " + rand.nextFloat());
        System.out.println("Long random number: " + rand.nextLong());
    }
}
```

```
Integer random number: -239054193
Integer random number from 0 to 50 is: 17
Boolean random value: true
Double random number: 0.711081415612728
Float random number: 0.2067644
Long random number: -8366989574191013407
```

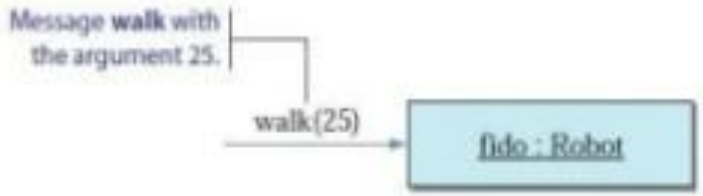
Motivate


We fall. We fail. We break.
But then, we rise. We heal.
We overcome.
@successpictures

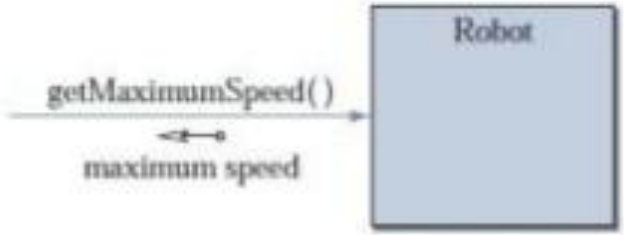
Motivate

Class and Instance Methods

- Instance Method : a method defined for an object.



- Class Method : a method defined for a class.



*an introduction to object oriented programming in java ,5th Edition , C . Thomas WU