Introduction

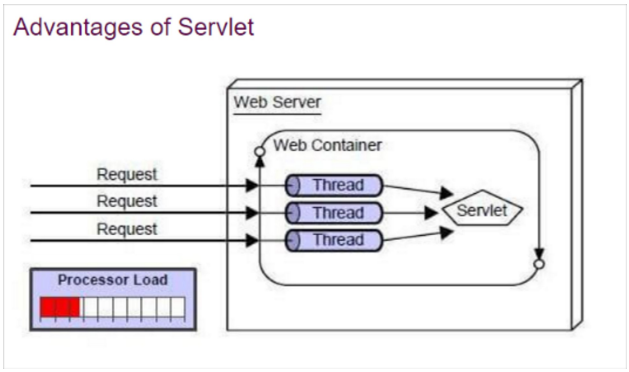
Tuesday, May 19, 2020 1:25 PM

What is a Servlet

- Servlet technology is used to create web applications.
- Servlet API provides **interfaces and classes** used to create web application.
- Using Servlets allows the JVM to **handle each request within a separate Java thread**, and this is one of the key advantage of Servlet container.
- **Each servlet is a Java class** with special elements responding to HTTP requests.

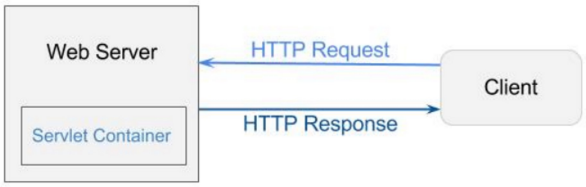
Benefits of Servlets

1. **Better performance:** because it creates a thread for each request, not process.
2. **Portability:** because it uses Java language.
3. **Robust:** JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
4. **Secure:** because it uses java language.



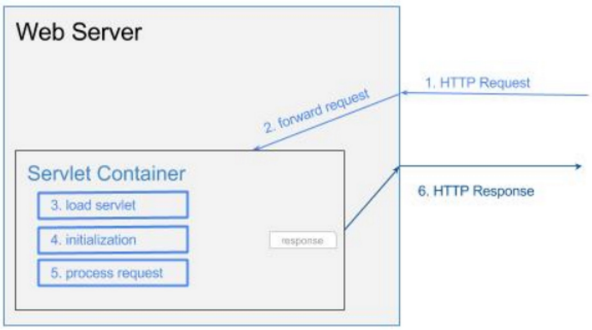
What is a Servlet Container?

- It's a part of web server that **interacts with the servlets**.
- It uses Java to d**ynamically generate web page** on server side.
- It helps **web server** to process a web request.
- The main function of Servlet contain is to **forward requests to correct servlet for processing**, and return the dynamically generated results to the correct location after the JVM has processed them.



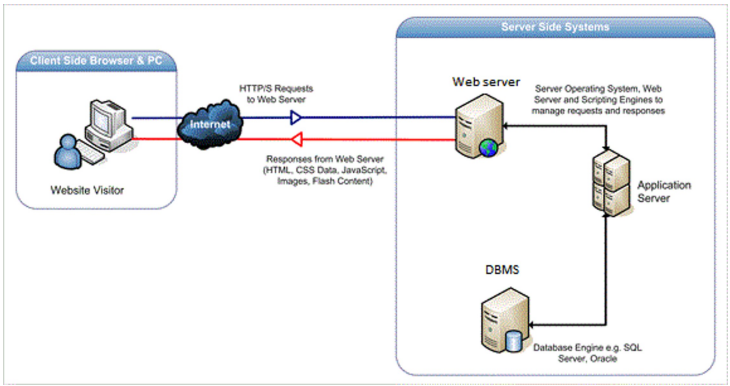
How does Servlet Container help Web Server ?

1. Web server receives the HTTP request
2. Web server forwards HTTP request to Servlet Container which is a part of Web Server.
3. Servlet Container retrieves and loads the Servlet in its address space.
4. Servlet Container invokes the init() method of Servlet.
5. Servlet Container invokes service() method of Servlet which processes the HTTP request.
6. The servlet remains in the container's address space and can process other HTTP requests.
7. Web server returns the dynamically generated response to the client.



Web Server vs Applications Server vs Web Container/Servlet Container

- Web Server
  - Its generally a server program which serves content using http/https protocol
  - Web container or Servlet container is part of Web Server.
  - It listens to incoming requests using HTTP protocol on a particular port.
  - Eg : Apache HTTPD is a Web Server.
- Web Container or Servlet Container
  - It is a part of web server.
  - Its used to manage the servlets and JSP etc.
  - Eg : Apache Tomcat - It's a web server + web container in one.
- Application Server
  - It's a heavyweight framework which handles application business logic.
  - It can interact with other back end applicataions of the organization.
  - It can interact with Database also. The functionality is inbuilt.
  - Eg IBM - WebSphere Application Server
  - Another Eg is a combination of Apache Tomcat + EJB container + APIs (JDBC, JNDI, JTA/JTS, JCA, JMX, JAAS, Java Mail, JMS).
- A web container runs only WARs, an application server runs EARs.



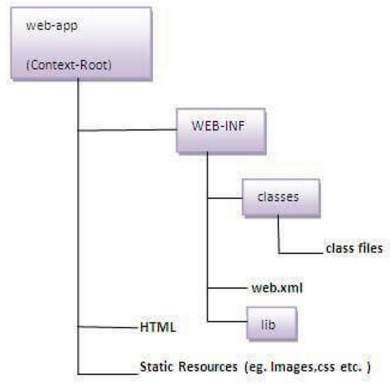
Servlet API

Tuesday, May 19, 2020 4:32 PM

Package

- javax.servlet --> Contains classes and interfaces used by Servlet and Servlet Container(Web Container) only.
- javax.servlet.http --> Classes and interfaces responsible for HTTP requests only.

Directory Structure



Creating Servlet using Eclipse IDE

1. Start a new Project in Eclipse --> **Dynamic Web Project**
2. Configure Tomcat Server in Eclipse before starting. Link in Important Links section.
3. Right click and 'New'->'Servlet'
4. Give it a name. Click Finish.
5. Start server.
6. Enter URL in browser as localhost:8080/projectname/webservletmappingname
7. See if your servlet is extending HTTPServlet class and @WebServlet annotations is present.

3 Ways of creating servlet

1. Implement **Servlet interface.**
2. Extend **GenericServlet class**
3. Extend **HTTPServlet class** --> This is the most common.

This is because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.

Deployment Descriptor

Before Servlet 3.0 , A web.xml file which maps URL to Servlets was required for the web container to determine which servlet to call on encountering a URL.
But starting from Servlet 3.0, **there is no need of web.xml**. Annotations do that job like @WebServlet.

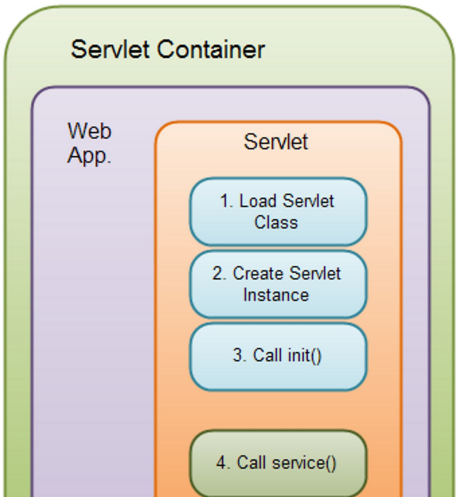
The web.xml file is replaced by following annotations available in **javax.servlet.annotation** package.
But you should have tomcat7 as it will not run in the previous versions of tomcat.

Annotation Types Summary	
HandlesTypes	This annotation is used to declare the class types that a ServletContainerInitializer can handle.
HttpConstraint	This annotation is used within the ServletSecurity annotation to represent the security constraints to be applied to all HTTP protocol methods for which a corresponding HttpMethodConstraint element does NOT occur within the ServletSecurity annotation.
HttpMethodConstraint	This annotation is used within the ServletSecurity annotation to represent security constraints on specific HTTP protocol messages.
MultipartConfig	Annotation that may be specified on a Servlet class, indicating that instances of the Servlet expect requests that conform to the multipart/form-data MIME type.
ServletSecurity	This annotation is used on a Servlet implementation class to specify security constraints to be enforced by a Servlet container on HTTP protocol messages.
WebFilter	Annotation used to declare a servlet filter.
WebInitParam	This annotation is used on a Servlet or Filter implementation class to specify an initialization parameter.
WebListener	This annotation is used to declare a WebListener.
WebServlet	Annotation used to declare a servlet.

Java Servlet Life Cycle

A servlet follows a certain life cycle. The servlet life cycle is managed by the servlet container. The life cycle contains the following steps:

1. Load Servlet Class.
2. Create Instance of Servlet.
3. Call the servlets init() method.
4. Call the servlets service() method.
5. Call the servlets destroy() method.



For every request received to the servlet, the servlets service() method is called. For HttpServlet subclasses, one of the doGet(), doPost() etc. methods are typically called.

Servlet Classes and Interface

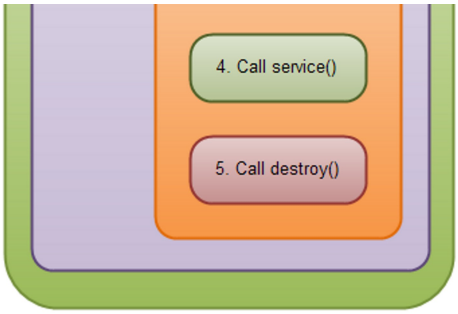
Servlet Interface

There are 5 methods in Servlet interface :-

Servlet Interface

There are 5 methods in Servlet interface :-

Method	Description
public void init(ServletConfig config)	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
public void service(ServletRequest request,ServletResponse response)	Provides response for the incoming request. It is invoked at each request by the web container.
public void destroy()	Is invoked only once and indicates that servlet is being destroyed.
public ServletConfig getServletConfig()	returns the object of ServletConfig.
public String getServletInfo()	returns information about servlet such as writer, copyright, version etc.



GenericServlet Class

Implements **Servlet, ServletConfig and Serializable**

There are many methods in GenericServlet class. They are as follows:

1. **public void init(ServletConfig config)** is used to initialize the servlet.
2. **public abstract void service(ServletRequest request, ServletResponse response)** provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. **public ServletConfig getServletConfig()** returns the object of ServletConfig.
5. **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.
6. **public void init()** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
7. **public ServletContext getServletContext()** returns the object of ServletContext.
8. **public String getInitParameter(String name)** returns the parameter value for the given parameter name.
9. **public Enumeration getInitParameterNames()** returns all the parameters defined in the web.xml file.
10. **public String getServletName()** returns the name of the servlet object.
11. **public void log(String msg)** writes the given message in the servlet log file.
12. **public void log(String msg,Throwable t)** writes the explanatory message in the servlet log file and a stack trace.

HttpServlet class

1. **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
4. **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
5. **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.
6. **protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.
7. **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.
8. **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.
9. **protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.
10. **protected long getLastModified(HttpServletRequest req)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

How to create war file?

To create war file, you need to use **jar tool** of JDK. You need to use **-c** switch of jar, to create the war file.

Go inside the project directory of your project (outside the WEB-INF), then write the following command:

```
jar -cvf projectname.war *
```

Here, **-c** is used to create file, **-v** to generate the verbose output and **-f** to specify the arhive file name.

The *** (asterisk) symbol** signifies that all the files of this directory (including sub directory).

Load on startup

```
<servlet>
 <servlet-name>servlet1</servlet-name>
 <servlet-class>com.javatpoint.FirstServlet</servlet-class>
 <load-on-startup>0</load-on-startup>
</servlet>
```

- As servlet is initialized on the first request, it takes more time for first request to complete. That's why we use load on startup to automatically load servlets at the time of server start or deployment. The no. in load--on-startup determines the order of loading of servlets in the servlet container.

ServletContext

The ServletContext is an object that contains meta informaton about your web application. You can access it via the HttpRequest object, like this:

```
ServletContext context = request.getSession().getServletContext();
context.setAttribute("someValue", "aValue");
Object attribute = context.getAttribute("someValue");
```

Servlet Code

Wednesday, May 20, 2020 12:54 AM

```
package sec;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class Hello
 */
@WebServlet("/Hello")
public class Hello extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * Default constructor.
     */
    public Hello() {
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request,
    HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        response.getWriter().append("Served at:
        ").append(request.getContextPath());
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request,
    HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }

}
```

Important Links

Tuesday, May 19, 2020 11:46 PM

Installing Tomcat on Windows PC	https://www.examtray.com/tutorials/how-install-tomcat-server-on-windows-10-8-and-7
Creating Servlets using Eclipse IDE & Tomcat	https://www.javatpoint.com/steps-to-create-a-servlet-using-tomcat-server
Is WEB.xml needed?	https://stackoverflow.com/questions/30259153/do-i-really-need-web-xml-for-a-servlet-based-java-web-application
web.xml Servlet Configuration	http://tutorials.jenkov.com/java-servlets/web-xml.html
Steps to Create Servlet Application using tomcat server	https://www.studytonight.com/servlet/steps-to-create-servlet-using-tomcat-server.php
Java Servlet POST Example	https://javatutorial.net/java-servlet-post-example

Generics

Thursday, May 21, 2020 11:56 PM

Why use Generics?

- It helps us turn runtime errors into compile time errors. We do this by removing the need of explicit casting.
- This is very beneficial as compile time errors are very easy top identify and fix.
- Using generics helps us in reducing the number of methods to be written while passing parameters of different types ie. We can define one method with generic type instead of multiple.
- Thus, using generics promotes code re-usability.
- Another, benefit is usage of generic algorithms becomes a viable option for developers.

Instead of

Public void draw(Car car) or
Public coid draw(Bus bus)

We can use
Public void draw(T t)
Thus code is reduced.

```
1  package app;
2
3  class MyClass {
4
5      public Object obj ;
6
7      public Object getObj() {
8          return this.obj;
9      }
10
11     public void setObj(Object o) {
12         this.obj = o ;
13     }
14
15 }
16
17 public class myApp {
18
19     public static void main(String args[]) {
20         MyClass m = new MyClass();
21         m.setObj(45);
22         Integer i = (Integer) m.getObj() ;
23         System.out.println(i);
24     }
25
26 }
```

We can always use generic class Object as type of Generic variable as required.

! The problem here is that the object needs to be cast into some Type for retrieving. This is not cool as we need to cast the object so we need to know its type exactly. Also, casting is expensive for performance.

This is explicit casting

That's why we proceed with using Generics.

```
1  package app;
2
3  class MyClass<T> {
4
5      public T obj ;
6
7      public T getObj() {
8          return this.obj;
9      }
10
11     public void setObj(T o) {
12         this.obj = o ;
13     }
14
15 }
16
17 public class myApp {
18
19     public static void main(String args[]) {
20         MyClass<Integer> m = new MyClass<>();
21         m.setObj(45);
22         Integer i =  m.getObj() ;
23         System.out.println(i);
24     }
25
26 }
```

We are using the Diamond Operator <> to specify a general type T

Under the hood, this T is replaced by whatever type that we pass while creating object reference.

The T can be replaced by any name of our choice.

While creating an instance, We have to again use Diamond Operator<> to pass in the type of generic object that we have to create

As casting is not required in this case, our main objective is fulfilled.

As we are not going to be casting, the casting (under the hood in Generics) is going to happen in compile time, so run time bugs and defects will get reduced. Explicit casting does casting in run time.

Multiple Type Parameters

```
17 class HashTable<K,V> {
18
19     public K key ;
20     public V value;
21
22     public HashTable(K key, V val) {
23         this.key = key ;
```

Two different parameters passed.

```java
class HashTable<K,V> {

    public K key ;
    public V value;

    public HashTable(K key, V val) {
        this.key = key ;
        this.value = val ;
    }

    @Override
    public String toString() {
        return "HashTable [key=" + key + ", value=" + value + "]";
    }


}

public class myApp {

    public static void main(String args[]) {
        HashTable<Integer, String> ht = new HashTable<>(1, "Damn this world");
        System.out.println(ht.toString());
    }

}
```

Two different parameters passed.

Values passed to constructor.

No need to pass types here again. Its OK to keep blank.

```
Problems  Javadoc  Declaration  Console
<terminated> myApp [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (May 22, 2020, 12:43:19 PM
HashTable [key=1, value=Damn this world]
```

Generic Methods I

```java
package app;

public class GenClass {

    // We are defining a generic method here.

    public <T> void genericMethod1(T t) {
        System.out.println(t);
    }

    public <T> T genericMethod2(T t) {
        return t ;
    }
}
```

Need not pass in generic types here as this class does not have generic properties of its own. The generic types are just parameters to methods. Those will be provided by calling class.

Even if the method's return type is void, we have to specify that the method is a generic method. That is why we specify Diamond Operator <T> , It has got nothing to do with return parameters.

And when actually returning some value, we have to specify its type. In this case it will be T.

```java
public class myApp {

    public static void main(String args[]) {
        GenClass gn = new GenClass();

        // Make call to generic methods
        gn.genericMethod1("Some String");
        System.out.println(gn.genericMethod2(8888));
    }
```

Pass in parameter of any type.

Generic Methods II

```java
package app;

public class GenClass {

    // We are defining a generic method here.

    public <T> void genericMethod1(T t) {
        System.out.println(t);
    }

    public <T> T genericMethod2(T t) {
        return t ;
    }
```

Generic array is defined as T[]

```
    public <T> T genericMethod2(T t) {
        return t ;
    }

    // Method to iterate array of generic type
    public <T> void showArray(T[] t) {
        for(T item : t) {
            System.out.println(item + "-->");
        }
    }
}
```

For each loop used to iterate generic array.

```
public class myApp {

    public static void main(String args[]) {
        GenClass gn = new GenClass();

        String[] stringArray = {"Piyush", "Mangesh", "Dangre"};
        Integer[] intArray = {1,2,3,4,5} ;

        gn.showArray(stringArray);
        gn.showArray(intArray);

    }

}
```

Pass in array of whatever type.

Bounded Parameter Type

We can limit the range of types to be passed as parameters to Generic method. This can be done by specifying the interface or class which has been inherited by the particular type.

The program will compare any two types which implement Comparable interface.

```
class Person implements Comparable {

    @Override
    public int compareTo(Object arg0) {
        // TODO Auto-generated method stub
        return 0;
    }

}

public class myApp {

    public static <T extends Comparable> T compare(T t1, T t2) {

        if(t1.compareTo(t2) < 0) {
            return t1 ;
        }

        else
            return t2 ;
    }

    public static void main(String args[]) {
        System.out.println(compare(20,19));
        Person p1 = new Person();
        Person p2 = new Person();
        System.out.println(compare(p1,p2));

    }

}
```

Person implements Comparable

Here it is being specified that whatever type parameter that will be specified, it has to implement the comparable interface. Notice that the interface is always 'implemented' but here we are using keyword 'extends'.

Strings, Integers , Double etc. implement the Comparable interface.

What happens if Person class does not implement Comparable interface?

```java
class Person {

}

public class myApp {

    public static <T extends Comparable> T compare(T t1, T t2) {

        if(t1.compareTo(t2) < 0) {
            return t1 ;
        }

        else
            return t2 ;
    }

    public static void main(String args[]) {
        System.out.println(compare(20,19));
        Person p1 = new Person();
        Person p2 = new Person();
        System.out.println(compare(p1,p2));

    }

}
```

If the Person class does not implement Comparable interface, a compiler error is thrown. Thus we are able to keep the parameters into bound.

Type Inference

```java
class Person implements Comparable {

    @Override
    public int compareTo(Object arg0) {
        // TODO Auto-generated method stub
        return 0;
    }

}

public class myApp {

    public static <T extends Comparable> T compare(T t1, T t2) {

        if(t1.compareTo(t2) < 0) {
            return t1 ;
        }

        else
            return t2 ;
    }

    public static void main(String args[]) {
        System.out.println(compare(20,19));
        Person p1 = new Person();
        Person p2 = new Person();
        System.out.println(myApp.<Person>compare(p1,p2));

    }

}
```

We can also specify the method with its full signature, as to what Type are passing to it. Generally, this is not required as Java automatically infers the type by a process called type inference.

Wildcards

Friday, May 22, 2020 3:16 PM

What is the need of Wildcards?

In Java , The Object class is the parent of all classes, like the String for example.
So we can pass in String object to a method which accepts Object type parameter.

This accepts all kind of arguments.

```
65    public void acceptAll(Object obj) {
66        System.out.println(obj.toString());
67    }
```

But if we want to pass in Collection of Strings to a method accepting collection of Object , Then that will not work in java.

That means String is an Object in Java but a Collection<String> is not Collection<Object>

```
60    public static void main(String args[]) {
61
62        Collection<String> c = new ArrayList<>();
63        myApp.acceptAll(c);
64
65    }
66
67    public static void acceptAll(Collection<Object> obj) {
68        System.out.println(obj.toString());
69    }
```

Throws compile time error.

This is where Wildcard is needed.

Wildcard acts like a Super type of collections of All types.

We use the <?> notation to describe wildcard. As done below --

```
60    public static void main(String args[]) {
61
62        Collection<String> c = new ArrayList<>();
63        c.add("Piyush");
64        myApp.acceptAll(c);
65
66    }
67
68    public static void acceptAll(Collection<?> obj) {
69        System.out.println(obj.toString());
70    }
71
```

This now accepts collection of any type.

One catch is of Wildcard -->

We cannot add/increment element in the collection when we use it within the method having wildcard parameter.

Two types of WildCards

- Bounded Wildcards
- Unbounded Wildcards

This is an example of Unbounded Wildcard as there is no parent class to object in collection. There is no keyword as Extends or Super.

```
60    public static void main(String args[]) {
61
62        Collection<String> c = new ArrayList<>();
63        c.add("Piyush");
64        myApp.acceptAll(c);
65
66    }
67
68    public static void acceptAll(Collection<?> obj) {
69        System.out.println(obj.toString());
70    }
71
```

```
package app;

import java.util.ArrayList;
import java.util.Collection;

public class Application {

    public static void main(String[] args) {
        Collection<Fruit> fruits = new ArrayList<>();
        fruits.add(new Mango());
        fruits.add(new Apple());
        Application.eat(fruits);
    }

    public static void eat(Collection<? extends Fruit> fruits) {
        for(Fruit frut : fruits) {
            System.out.println(frut);
        }
    }
```

This is Bounded Wildcards where a parent class is specified with extends or super keyword.

If we do not use the extends keyword, then there is compile time error in for loop.

This error is 'Type Mismatch' as Java does not know what is the type of the single element being iterated.

```
        for(Fruit frut : fruits) {
            System.out.println(frut);
        }
    }
}

class Fruit {
    String taste;
    Fruit(){}
    Fruit(String taste){
        this.taste = taste;
    }
}

class Apple extends Fruit {   }

class Mango extends Fruit {   }
```

time error in for loop.

This error is 'Type Mismatch' as Java does not know what is the type of the single element being iterated.

So, although, the method accepts the parameter, we cannot process the collection (through loop) as there is compile time error. That is why extends is used to bound the parameter to a certain parent class.

Here, we can iterate the fruits list using only Fruit reference. We cannot use subclass like Apple or Mango for containing each element in collection.

But we cannot add element into the collection using this method.

As shown in below code.

```
15    public static void eat(Collection<? extends Fruit> fruits) {
16        for(Fruit frut : fruits) {
17            System.out.println(frut);
18        }
19
20        fruits.add(new Fruit());
21        fruits.add(new Apple());
22    }
23 }
```

Throws compile time error. So what should be done to add into Collection?

To add an element in the collection , use super keyword

- This is done primarily to add an element in the collection.
- Reading the element is not guaranteed. We have to cast the element into Object superclass.

```
 6 public class Application {
 7
 8     public static void main(String[] args) {
 9         Collection<Fruit> fruits = new ArrayList<>();
10         fruits.add(new Mango());
11         fruits.add(new Apple());
12         Application.eat(fruits);
13     }
14
15     public static void eat(Collection<? super Fruit> fruits) {
16         for(Object frut : fruits) {
17             System.out.println(frut);
18         }
19
20         fruits.add(new Fruit());
21         fruits.add(new Apple());
22
23         for(Object frut : fruits) {
24             System.out.println(frut);
25         }
26     }
27 }
28
29 class Fruit {
30     String taste;
31     Fruit(){}
32     Fruit(String taste){
33         this.taste = taste;
34     }
35 }
36
37 class Apple extends Fruit { }
38
39 class Mango extends Fruit { }
```

Using super to add into the collection.

Casting into Object super class while reading from the collection while iterating.

So there are two rules -->

1. To add into collection use Super keyword.
2. To read from collection only, use Extends keyword.

Addition into generic list

```
15        List<? extends Number> list = new ArrayList<Integer>();
16        list.add(new Integer(5));
```

Using extends lets us declare the arraylist but addition of integer throws error.

```
15        List<? super Number> list = new ArrayList<Integer>();
16        list.add(new Integer(5));
```

Using super , Java lets us add element into the collection. This is because we have declared the type of objects inside the List to be of super types of Number. This includes Object, Serializable etc. Since Object is allowed, it allows Integer, Float as well since all objects in java are Object class.

```
15        List<? super Number> list = new ArrayList<Object>();
16        list.add(new Integer(5));
17
```

We have to store the ArrayList to be of Object element. Otherwise throws error.

```
15        List<? super Number> list = new ArrayList<>();
16        list.add(new Integer(5));
```

Or do not use anything to declare ArrayList, that's OK.

Best Explanation --> http://tutorials.jenkov.com/java-generics/wildcards.html

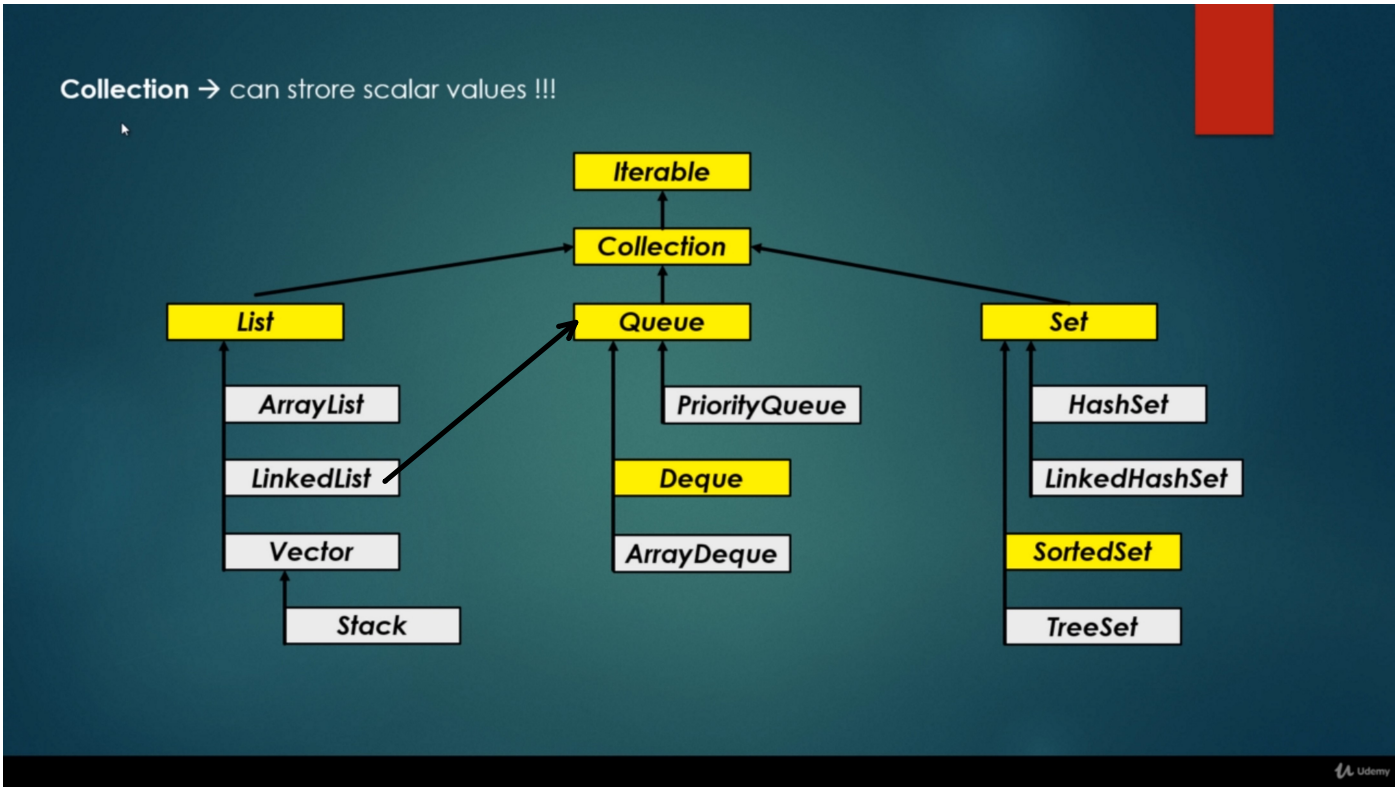
Code with comments for explanation -->

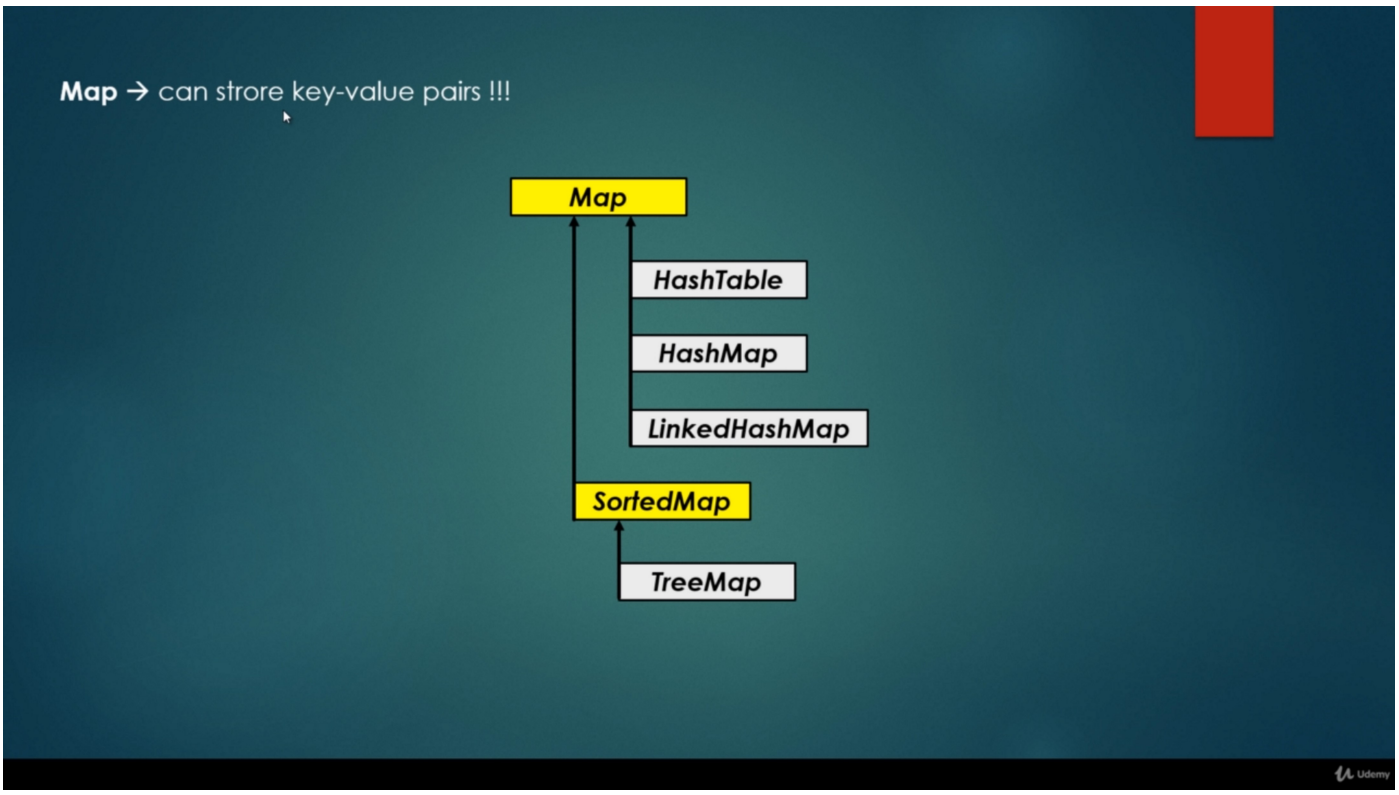
https://github.com/PiyushDangre/Core-Java/blob/master/GenericPract/src/app/Generics.java

If we want to add and read at the same time from the list (read without casting to object) , we have to declare the collection with a specific type like List<Double>.

Collections

Saturday, May 23, 2020 5:23 PM





Collection Interface extends Iterable interface.

ArrayList

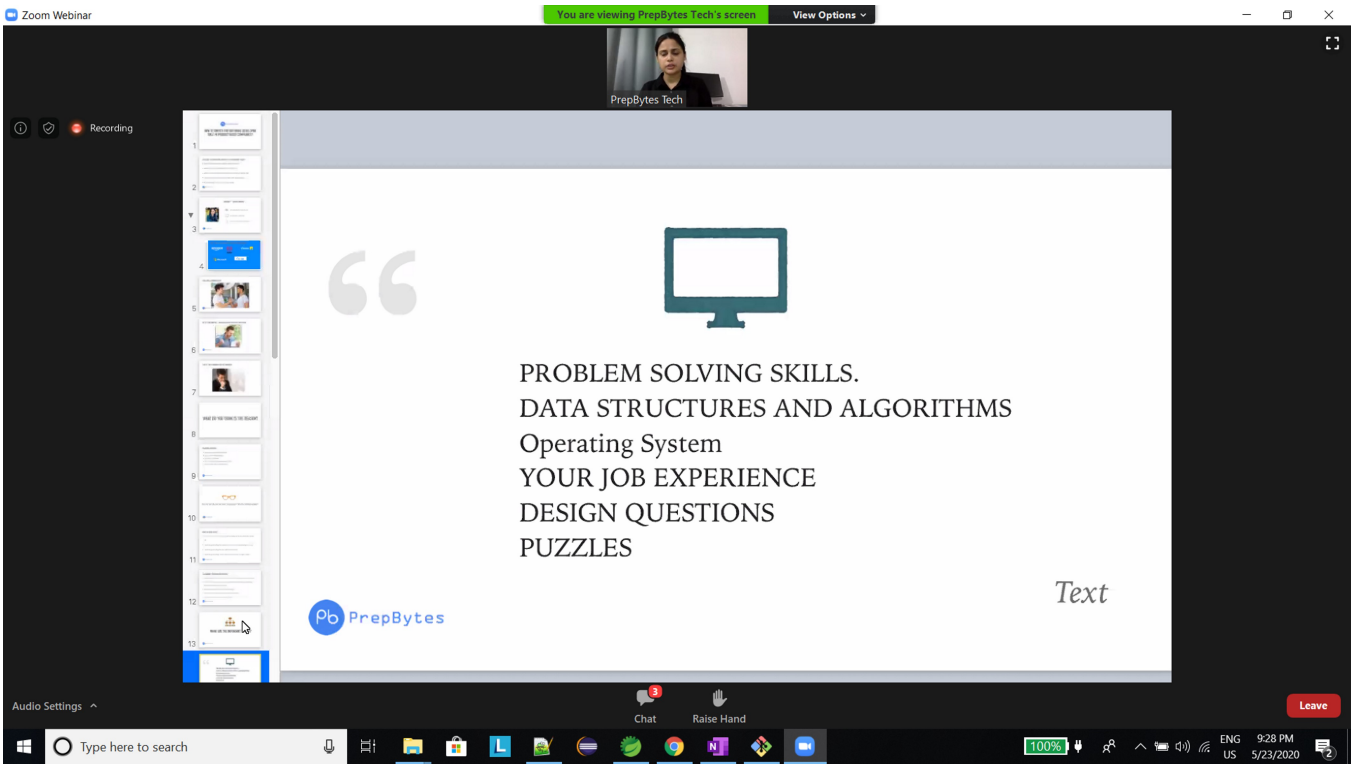
1. ArrayList stores the values in an array under the hood
2. An initial capacity can be defined in the arraylist.
3. Arraylist is very fast in data retrieval due to random access(arr.get(i))
4. Arraylist is slow in element removal operation as there needs to be lot of restructuring of other elements.

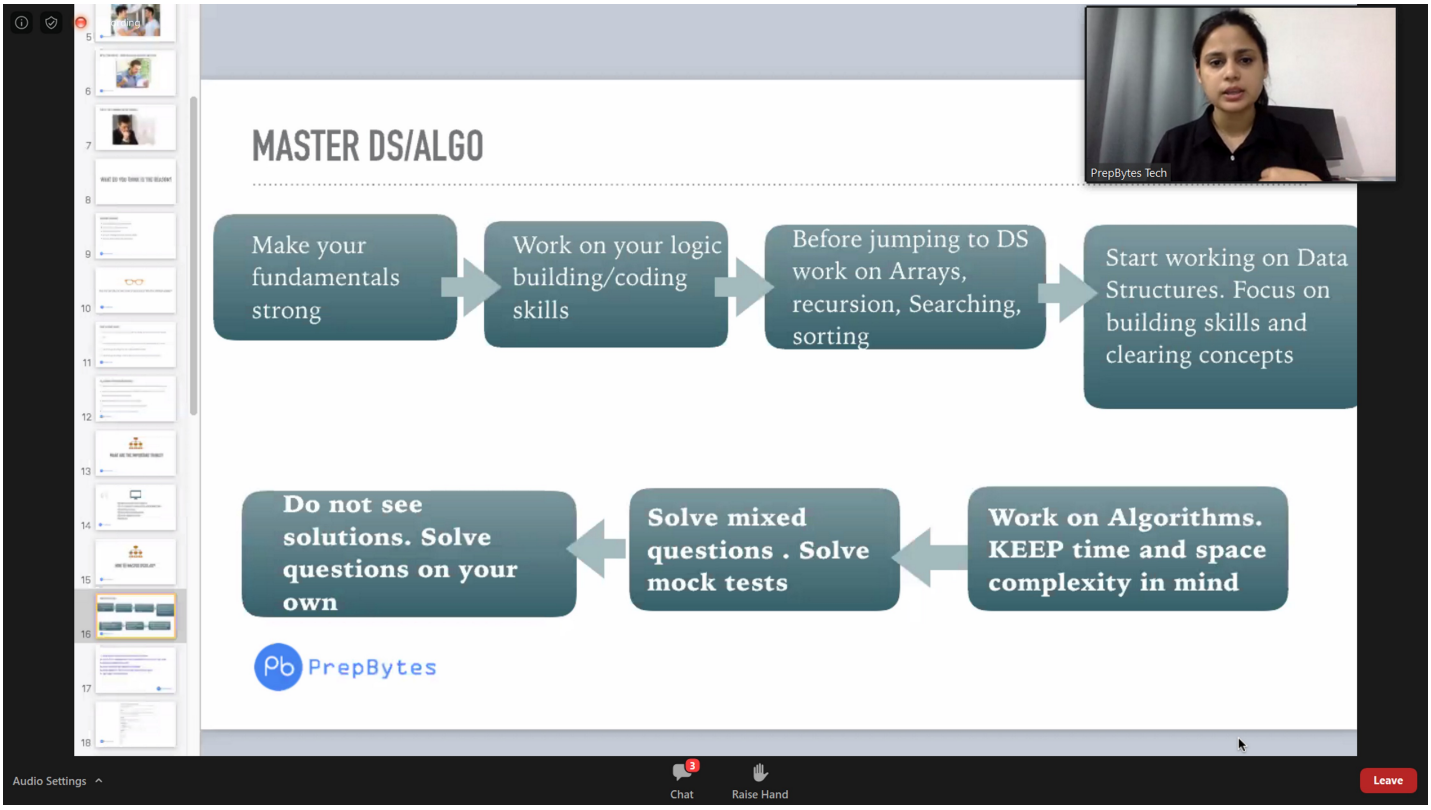
Priority Queue

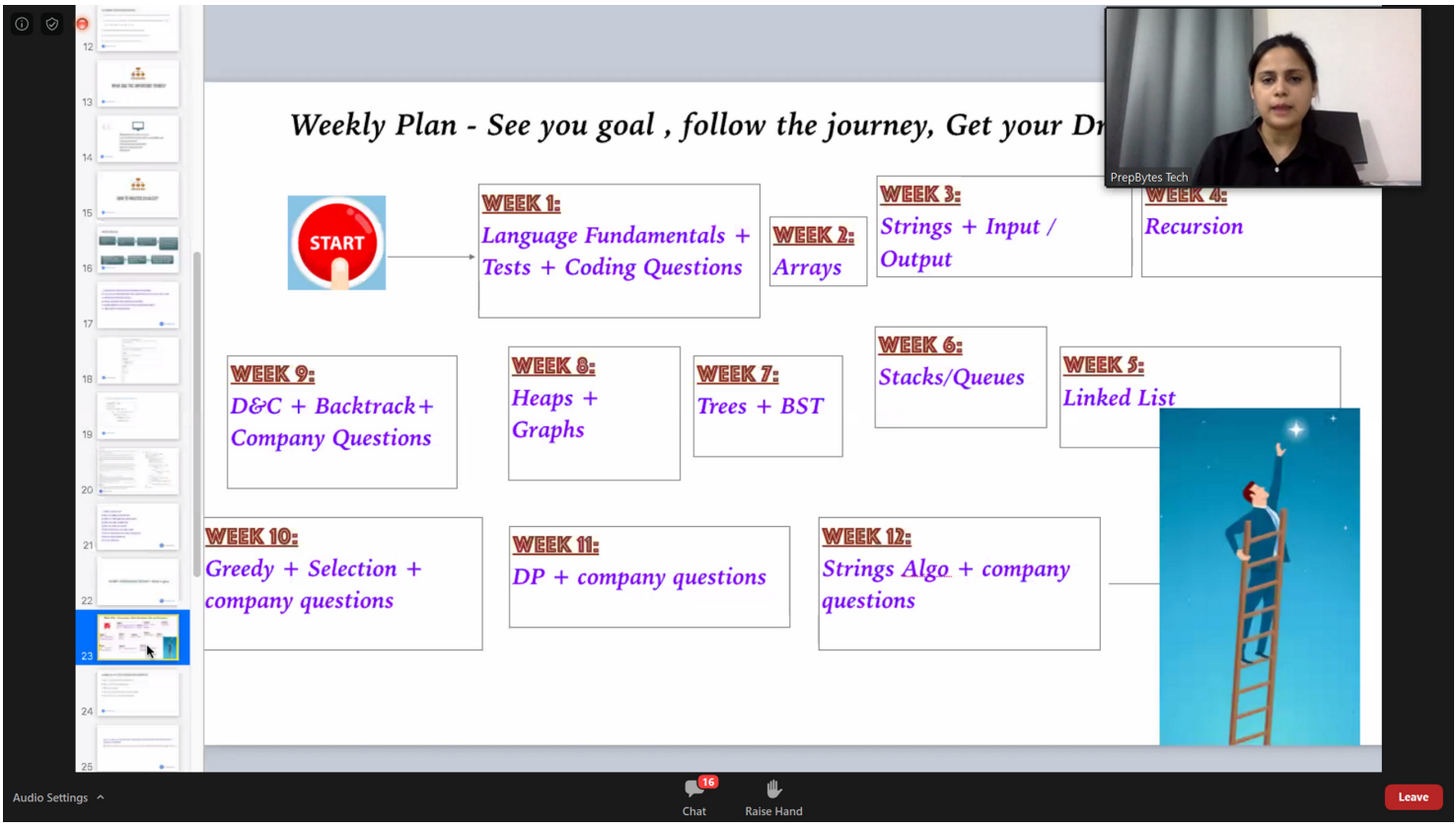
1. Priority Queue is concrete implementation of Queue
2. In this queue, the elements are sorted by some property.
3. This property (or member variable) is defined in the compareTo method of class element.
4. Every element's class has to implement Comparable interface.
5. Due to implementation of Comparable interface, they have to mention the sorting condition in the CompareTo method.
6. Peek() --> Returns the head of the queue
7. Poll() --> Returns and **removes** the head of the queue.
8. In this queue, we have to specify each nodes priority. According to what property does each node has to be prioritized? That's why we implement Comparable interface.

Deque

1. It stands for doubly ended queue.
2. ArrayDeque is the concrete implementation of Deque.\
3. We can insert and remove elements from both sides of the queue.
4. It can be used both as a stack and a queue.
5. When used a stack , its faster than stack.
6. When used as queue, its faster than LinkedList.
7. offerFirst(T t) --> Insert in the starting of the queue.
8. Offerlast(T t) --> Insert in the end of the queue.
9. removeLast() and removeFirst() methods are also provided.
10. Its not used much in software world as it is too complex data structure.



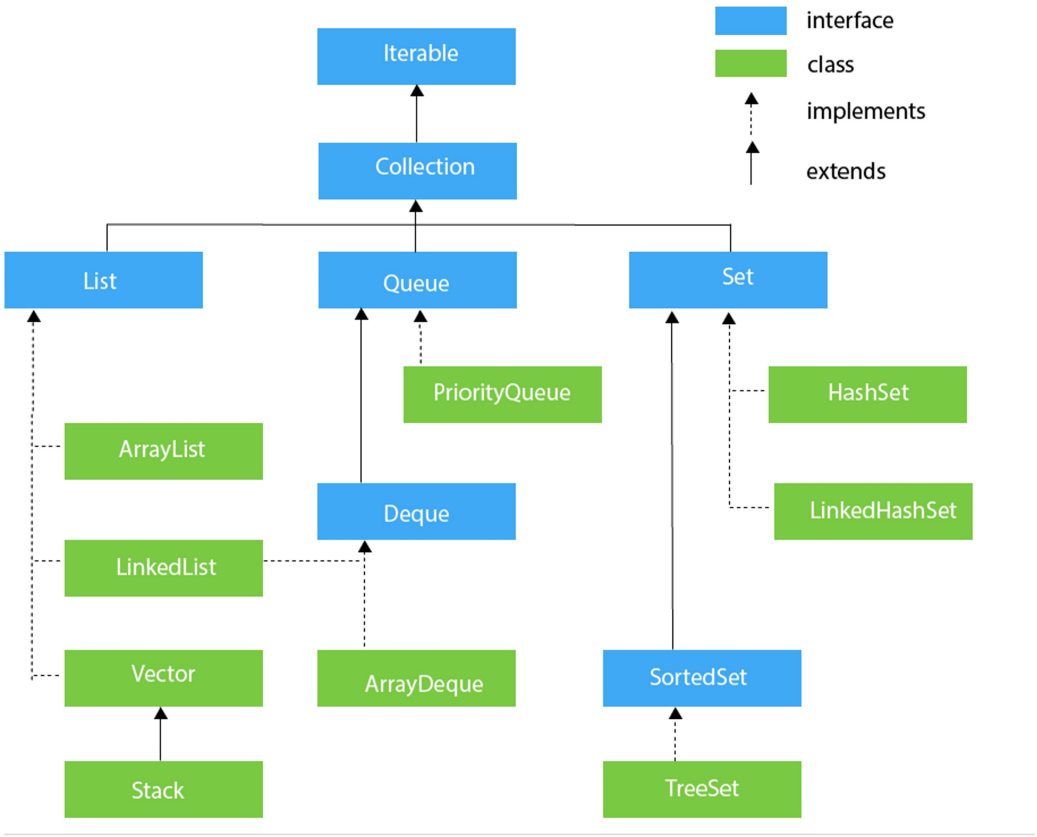

MASTER DS/ALGO
Make your fundamentals strong
Work on your logic building/coding skills
Before jumping to DS work on Arrays, recursion, Searching, sorting
Start working on Data Structures. Focus on building skills and clearing concepts
Work on Algorithms. KEEP time and space complexity in mind
Solve mixed questions . Solve mock tests
Do not see solutions. Solve questions on your own
PrepBytes
PrepBytes Tech
Audio Settings
Chat
Raise Hand
Leave


Weekly Plan - See you goal , follow the journey, Get your Dr
START
WEEK 1:
Language Fundamentals + Tests + Coding Questions
WEEK 2:
Arrays
WEEK 3:
Strings + Input / Output
WEEK 4:
Recursion
WEEK 5:
Linked List
WEEK 6:
Stacks/Queues
WEEK 7:
Trees + BST
WEEK 8:
Heaps + Graphs
WEEK 9:
D&C + Backtrack+ Company Questions
WEEK 10:
Greedy + Selection + company questions
WEEK 11:
DP + company questions
WEEK 12:
Strings Algo + company questions
PrepBytes Tech
Audio Settings
Chat
Raise Hand
Leave

Features

Monday, May 25, 2020 12:59 PM

Features of all collections



ArrayList

- Duplicates allowed.
- Maintains insertion order.
- Non synchronized
- Allows random access so fast reading.
- Manipulation is slow because of lot of shifting of elements.
- Implements simple Array structure under the hood.

https://github.com/PiyushDangre/JavaBasics/blob/master/src/collections/ListInterface.java

HashSet

- No Duplicates allowed.
- Insertion order not maintained. Ordered by hash value.
- Mechanism called hashing used to store elements.
- Non synchronized.
- Best for search operations.
- Allows only one null value.

https://github.com/PiyushDangre/JavaBasics/blob/master/src/collections/SetInterfaces.java

LinkedHashSet

- No duplicates allowed.
- **Maintains insertion order.**
- Up to one null value allowed.
- Non synchronized.

LinkedList

- Duplicates allowed
- Maintains insertion order
- Non synchronized
- Manipulation is fast because, no shifting of elements required.
- Implements doubly linked list under the hood.
- Implements both List and Deque interfaces
- Can be used as List, Stack or Queue.

In the case of a doubly linked list, we can add or remove elements from both sides.

fig- doubly linked list

We have access to following methods -

- addFirst()
- addLast()
- offer(E e) --> Append element at the end of list
- offerLast(E e)
- offerFirst(E e)
- Peek() --> Get the first element of the list
- Poll() --> gets and removes the first element of list
- Push() --> pushes element into stack represented by list.
- Pop() --> pops element into stack represented by list.
- descendingIterator() --> gets iterator object for reverse iteration.

https://github.com/PiyushDangre/JavaBasics/blob/master/src/collections/ListInterface.java

https://github.com/PiyushDangre/JavaBasics/blob/master/src/collections/SetInterfaces.java

TreeSet

- No duplicates allowed.
- **No null element allowed.**
- **Maintains natural ordering of elements.**
- Non synchronized.
- Access and retrieval times are fast. Due to Tree DS.

https://github.com/PiyushDangre/JavaBasics/blob/master/src/collections/TreeSetInterface.java

ArrayDeque

- Doubly ended queue.
- **Element removal and insertion at both ends.**
- **Null elements are not allowed**
- no capacity restrictions.
- ArrayDeque is faster than LinkedList and Stack.

Methods such as-
- offerFirst()
- pollLast()

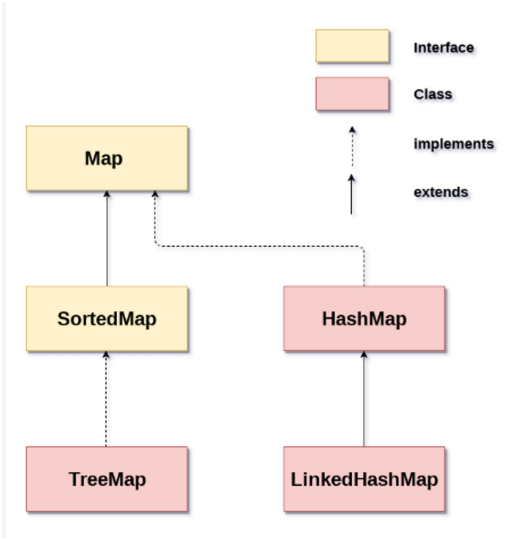
PriorityQueue

- Does not order in FIFO (Unlike plain Queue interface)
- Elements are ordered in Natural order or Comparator if present.
- **Null elements not allowed.**
- While the elements may not be arranged in natural order always , But they are always retrieved in natural order. For eg. The head is always replaced with lower or higher element according to the order defined in comparator.

Methods such as -
- Peek() - retrieves head and null if queue is empty
- Poll() - retrieves head and removes head, null if queue is empty
- Element() - gets head
- Remove() - gets and removes head.
- Offer(object) - Insert object In queue.

Good link - https://www.programiz.com/java-programming/priorityqueue
Code- https://github.com/PiyushDangre/JavaBasics/blob/master/src/collections/QueueInterface.java

Features of Map



What is Hashing?

Its is the process of converting an object into integer values.
The integer value helps in indexing and faster searches.
Internal working - https://www.javatpoint.com/working-of-hashmap-in-java

Map

- Stores key-value pairs
- Returns entrySet() in set form
- Returns keySet() and valueSet()
- Keys are unique always. Values can be different.

HashMap

- No duplicate keys.
- Duplicate values are allowed.
- Can contain one null key.
- Does not maintain any insertion order.
- Non synchronized

https://github.com/PiyushDangre/JavaBasics/blob/master/src/collections/MapInterface.java

TreeMap

- Maintains order based on natural ordering of key.
- Only unique keys.
- Cannot have null keys but can have null values.
- Non synchronized.

LinkedHashMap

- Similar like HashMap
- But maintains insertion order.

HashTable

- It is synchronized.