Q=how to provide priority in executer framework?

Q=dieference between data hiding and abstraction?

Ans=Data hiding is the process by which access modifiers are used to hide the visibility of java methods and variables. They access modifiers are: public, private and protected.

Abstraction is the process by which we define a specific behavior by beans of abstract classes and methods which form the skeleton for any class that would be extending this class

**Encapsulation** in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.

To achieve encapsulation in Java −

* Declare the variables of a class as private.
* Provide public setter and getter methods to modify and view the variables values.

Benefits of Encapsulation

* The fields of a class can be made read-only or write-only.
* A class can have total control over what is stored in its fields.

**abstraction** is a process of hiding the implementation details from the user, only the functionality will be provided to the user. In other words, the user will have the information on what the object does instead of how it does it.

In Java, abstraction is achieved using Abstract classes and interfaces.

## Abstract Class

A class which contains the **abstract** keyword in its declaration is known as abstract class.

* Abstract classes may or may not contain *abstract methods*, i.e., methods without body ( public void get(); )
* But, if a class has at least one abstract method, then the class **must**be declared abstract.
* If a class is declared abstract, it cannot be instantiated.
* To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it.
* If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

Q=query for find duplicate record in table?

Ans=SELECT

name, email, COUNT(\*)

FROM

users

GROUP BY

name, email

HAVING

COUNT(\*) > 1

Q=**Difference between WHERE and HAVING clause:**

**Ans:**

|  |  |
| --- | --- |
| down vote | **Difference between WHERE and HAVING clause:**  **1.** ***WHERE clause*** can be used with - Select, Insert, and Update statements, where as ***HAVING clause*** can only be used with the Select statement.  **2.** ***WHERE*** filters rows before aggregation (GROUPING), where as, ***HAVING*** filters groups, after the aggregations are performed.  **3.** Aggregate functions cannot be used in the ***WHERE clause***, unless it is in a sub query contained in a **HAVING clause**, whereas, aggregate functions can be used in Having clause.  **Filtering Groups:**  WHERE clause is used to filter rows before aggregation, where as HAVING clause is used to filter groups after aggregations  Select City, SUM(Salary) as TotalSalary  from tblEmployee  Where Gender = 'Male'  group by City  Having City = 'London'  In SQL Server we have got lot of aggregate functions. **Examples**   1. Count() 2. Sum() 3. avg() 4. Min() 5. Max() |

Q=get() method of e=executer framework is blocking how u use nonblocking?

Ans= we will use CompletableFuture class – introduced as a Java 8 Concurrency API

Q=how many thread pool we should create for n number of processors?

Ans= ExecutorService e = Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors());

// Do work using something like either

e.execute(new Runnable() {

public void run() {

// do one task

}

});

Q=when we will use int and when interger in java?

Ans= We sue wrapper classes in class lavel variable, because when class loads its default value will be null, so it will not consume any memory, but if we will use primitive type it will store in memory and its default value will load, and in method level variable we should use int primitive type.

Q=difference between 64bit java and 32bit java?

## Ans=Which versions of java you should install on 32-bit/64-bit machines?

Strictly speaking, on a 32-bit CPU architecture machine, you should install 32-bit java/JRE. On the other hand, on a 64-bit CPU architecture machine, you are free to choose between 32-bit java/JRE and 64-bit java/JRE. Both will work just fine. In fact, on 64-bit machine **decision of JRE version depends on other factors** such as maximum memory needed to run your application on high load scenarios.

Please note that high availability of memory doesn’t come for free. It does have a cost on runtime e.g.

**1)** 30-50% of more heap is required on 64-bit in comparison to 32-bit. Why? Mainly because of the memory layout in 64-bit architecture. First of all – object headers are 12 bytes on 64-bit JVM. Secondly, object references can be either 4 bytes or 8 bytes, depending on JVM flags and the size of the heap. This definitely adds some overhead compared to the 8 bytes on headers on 32-bit and 4 bytes on references.

**2)** Longer garbage collection pauses. Building up more heap means there is more work to be done by GC while cleaning it up from unused objects. What it means in real life is that you have to be extra cautious when building heaps larger than 12-16GB. Without fine tuning and measuring you can easily introduce full GC pauses spanning several minutes which can result in showstoppers.

Q=difference between shutdown and shutdownnow in java?

Ans=

In summary, you can think of it that way:

* shutdown() will just tell the executor service that it can't accept new tasks, but the already submitted tasks continue to run
* shutdownNow() will do the same AND will **try to cancel** the already submitted tasks by interrupting the relevant threads. Note that if your tasks ignore the interruption, shutdownNowwill behave exactly the same way as shutdown.

You can try the example below and replace shutdown by shutdownNow to better understand the different paths of execution:

* with shutdown, the output is Still waiting after 100ms: calling System.exit(0)... because the running task is **not** interrupted and continues to run.
* with shutdownNow, the output is interrupted and Exiting normally... because the running task is interrupted, catches the interruption and then stops what it is doing (breaks the while loop).
* with shutdownNow, if you comment out the lines within the while loop, you will get Still waiting after 100ms: calling System.exit(0)... because the interruption is not handled by the running task any longer.
* public static void main(String[] args) throws InterruptedException {
* ExecutorService executor = Executors.newFixedThreadPool(1);
* executor.submit(new Runnable() {
* @Override
* public void run() {
* while (true) {
* if (Thread.currentThread().isInterrupted()) {
* System.out.println("interrupted");
* break;
* }
* }
* }
* });
* executor.shutdown();
* if (!executor.awaitTermination(100, TimeUnit.MICROSECONDS)) {
* System.out.println("Still waiting after 100ms: calling System.exit(0)...");
* System.exit(0);
* }
* System.out.println("Exiting normally...");
* }

Q=how will u prove java is pass by value not reference?

Ans= We will pass the parameter and assign some other object so previous object will not change.

Ex:

# Java Pass-by-Value vs. Pass-by-Reference

March 6, 2013 by Lokesh Gupta

There has been a good amount of debate on whether “java is pass by value or pass by reference?”. Well, lets conclude it last time, **Java is pass by value and not pass by reference**. If it had been pass by reference, we should have been able to C like swapping of objects, but we can’t do that in java. We know it already, right?

When you pass an instance to a method, its memory address are copied bit by bit to new reference variable, thus both pointing to same instance. But if you change the reference inside method, original reference will not get change. If it was pass by reference, then it would have got changed also.

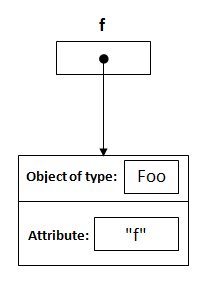
To prove it, lets see how memory allocations happen in run time. It should solve the slightest doubts, if any. I am using following program for demonstration of the concept.

|  |
| --- |
| public class Foo  {      private String attribute;        public Foo (String a){          this.attribute = a;      }      public String getAttribute() {          return attribute;      }      public void setAttribute(String attribute) {          this.attribute = attribute;      }  }    public class Main  {       public static void main(String[] args){            Foo f = new Foo("f");            changeReference(f); // It won't change the reference!            modifyReference(f); // It will change the object that the reference variable "f" refers to!       }       public static void changeReference(Foo a) {            Foo b = new Foo("b");            a = b;       }       public static void modifyReference(Foo c) {            c.setAttribute("c");       }  } |

Lets see what happen on runtime step by step :

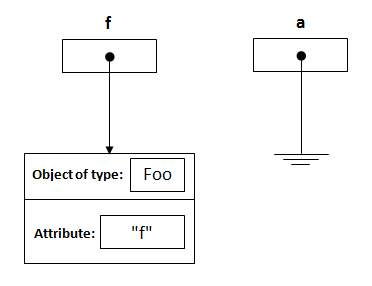
**1) Foo f = new Foo(“f”);**

This statement will create an instance of class Foo, with ‘attribute’ initialized to ‘f’. The reference to this created instance is assigned to variable f;

[](https://howtodoinjava.files.wordpress.com/2013/03/1.png)

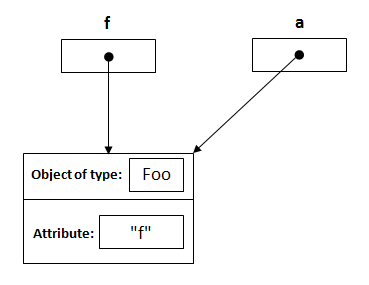
**2) public static void changeReference(Foo a)**

When this executes then a reference of type Foo with a name a is declared and it’s initially assigned to null.

[](https://howtodoinjava.files.wordpress.com/2013/03/2.png)

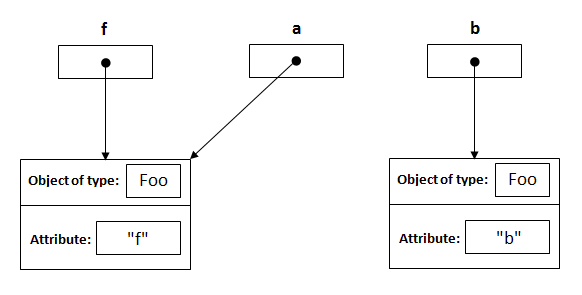
**3) changeReference(f);**

As you call the method changeReference, the reference a will be assigned to the object which is passed as an argument.

[](https://howtodoinjava.files.wordpress.com/2013/03/3.png)

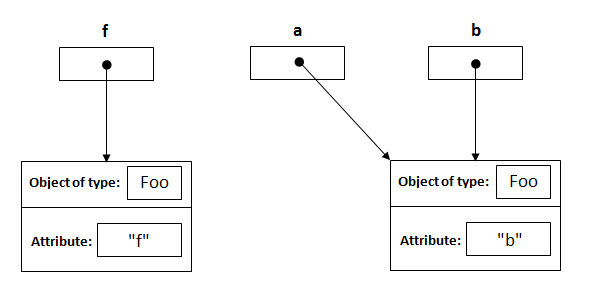
**4) Foo b = new Foo(“b”);** inside first method

This will do exactly the same as in first step, and will create a new instance of Foo, and assign it to b;

[](https://howtodoinjava.files.wordpress.com/2013/03/4.png)

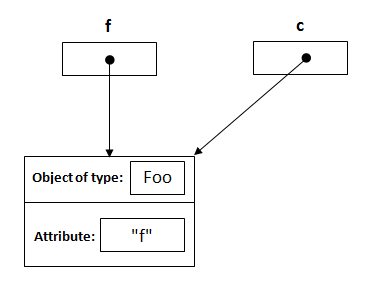
**5) a = b;**

This is the important point. Here, we have three reference variables and when statement executes, a and b will point to same instance created inside the method. Note: f is unchanged and it is continually pointing to instance, it was pointing originally. NO CHANGE !!

[](https://howtodoinjava.files.wordpress.com/2013/03/5.png)

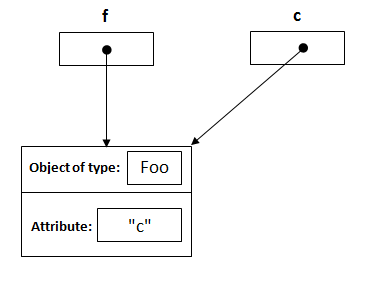
**6) modifyReference(Foo c);**

Now when this statement executed a reference, c is created and assigned to the object with attribute “f”.

[](https://howtodoinjava.files.wordpress.com/2013/03/6.png)

**7) c.setAttribute(“c”);**

This will change the attribute of the object that reference c points to it, and its same object that reference f points to it.

[](https://howtodoinjava.files.wordpress.com/2013/03/8.png)

Q= difference between fork join pool and executer service?

Q=Write an efficient program to count number of 1s in binary representation of an integer.?

Q=Given a positive integer, write a function to find if it is a power of two or not?

// Java program to find whether

// a no is power of two

import java.io.\*;

class GFG {

    // Function to check if

    // x is power of 2

    static boolean isPowerOfTwo(int n)

    {

        if (n == 0)

            return false;

        while (n != 1)

        {

            if (n % 2 != 0)

                return false;

            n = n / 2;

        }

        return true;

    }

    // Driver program

    public static void main(String args[])

    {

        if (isPowerOfTwo(31))

            System.out.println("Yes");

        else

            System.out.println("No");

        if (isPowerOfTwo(64))

            System.out.println("Yes");

        else

            System.out.println("No");

    }

}

Q=search a record from multiple file?

Q= design a metro card ?

Q= design a skype function in java?

Q=how to intract with two iframe?

# Ans=HTML JS communication between iframes and Cross-document messaging

Submitted by Anonymous (not verified) on Tue, 06/14/2016 - 17:46

**Question**

How to communicate between iframe and the parent site ?

How to communicate between two iframes  ?

Example :

For example in the top window (Parent): (**the\_iframe**.contentWindow.postMessage('data', 'domain');)

<**html**>  
  <**body**>  
    <**iframe** src="iframe.html" id="iframe1"></**iframe**>  
  </**body**>  
  <**script**>  
    window.onload = **function** () {  
      myIframe = document.getElementById('iframe1');  
      myIframe.contentWindow.postMessage('hello', '\*');  
    }  
  </**script**>  
</**html**>

and in the iframe:

<**html**>  
  <**body**>  
    <**div**>The I frame</**div**>  
  </**body**>  
  <**script**>  
    window.onmessage = **function** (e) {  
      **if** (e.data == 'hello') {  
        alert('It works!');  
      }  
    };  
  </**script**>  
</**html**>

On the **iframe**, You can also yse:  
function postMessage(**a**, b){  
}

Example 2 : Between two iframes

Get One iframe from another Iframe

**var** iframe2 = parent.document.getElementById('iframe2');

Parent:

<**html**>  
  <**body**>  
    <**iframe** src="iframe.html" id="iframe1"></**iframe**>  
    <**iframe** src="iframe2.html" id="iframe2"></**iframe**>  
  </**body**>  
</**html**>

iFrame 1:

<**html**>  
  <**script**>  
    window.onmessage = **function** (e) {  
        alert(e.data)  
    };  
  </**script**>  
</**html**>

iFrame2:

<**html**>  
  <**script**>  
    parent.window.onload = **function**() {  
      myIframe = parent.document.getElementById('iframe1');  
      myIframe.contentWindow.postMessage(["name", "age", "city"], '\*');  
    }  
  </**script**>  
</**html**>

Q=what is event bubbling?

Ans=An event received by an element doesn't stop with that one element. That event moves to other elements like the parent, and other ancestors of the element. This is called "event bubbling".

Q=difference between event bubbling and event capturing?

Ans=**Event capturing**

When you use event capturing

| |

---------------| |-----------------

| element1 | | |

| -----------| |----------- |

| |element2 \ / | |

| ------------------------- |

| Event CAPTURING |

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

the event handler of element1 fires first, the event handler of element2 fires last.

**Event bubbling**

When you use event bubbling

/ \

---------------| |-----------------

| element1 | | |

| -----------| |----------- |

| |element2 | | | |

| ------------------------- |

| Event BUBBLING |

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

the event handler of element2 fires first, the event handler of element1 fires last.

Event bubbling and capturing are two ways of event propagation in the HTML DOM API, when an event occurs in an element inside another element, and both elements have registered a handle for that event. The event propagation mode determines in [which order the elements receive the event](http://www.quirksmode.org/js/events_order.html).

With bubbling, the event is first captured and handled by the innermost element and then propagated to outer elements.

With capturing, the event is first captured by the outermost element and propagated to the inner elements.

Capturing is also called "trickling", which helps remember the propagation order:

trickle down, bubble up

Back in the old days, Netscape advocated event capturing, while Microsoft promoted event bubbling. Both are part of the W3C [Document Object Model Events](http://www.w3.org/TR/DOM-Level-2-Events/events.html) standard (2000).

IE < 9 uses [only event bubbling](https://developer.mozilla.org/en-US/docs/Web/API/EventTarget.addEventListener), whereas IE9+ and all major browsers support both. On the other hand, the [performance of event bubbling may be slightly lower](https://stackoverflow.com/a/10335117/1269037) for complex DOMs.

We can use the addEventListener(type, listener, useCapture) to register event handlers for in either bubbling (default) or capturing mode. To use the capturing model pass the third argument as true.

## Example

<div>

<ul>

<li></li>

</ul>

</div>

In the structure above, assume that a click event occurred in the li element.

In capturing model, the event will be handled by the div first (click event handlers in the div will fire first), then in the ul, then at the last in the target element, li.

In the bubbling model, the opposite will happen: the event will be first handled by the li, then by the ul, and at last by the div element.

For more information, see

* [Event Order](http://www.quirksmode.org/js/events_order.html) on QuirksMode
* [addEventListener](https://developer.mozilla.org/en-US/docs/Web/API/EventTarget.addEventListener) on MDN
* [Events Advanced](http://www.quirksmode.org/js/events_advanced.html) on QuirksMode

In the example below, if you click on any of the highlighted elements, you can see that the capturing phase of the event propagation flow occurs first, followed by the bubbling phase.

var logElement = document.getElementById('log');

function log(msg) {

logElement.innerHTML += ('<p>' + msg + '</p>');

}

function capture() {

log('capture: ' + this.firstChild.nodeValue.trim());

}

function bubble() {

log('bubble: ' + this.firstChild.nodeValue.trim());

}

var divs = document.getElementsByTagName('div');

for (var i = 0; i < divs.length; i++) {

divs[i].addEventListener('click', capture, true);

divs[i].addEventListener('click', bubble, false);

}

p {

line-height: 0;

}

div {

display:inline-block;

padding: 5px;

background: #fff;

border: 1px solid #aaa;

cursor: pointer;

}

div:hover {

border: 1px solid #faa;

background: #fdd;

}

<div>1

<div>2

<div>3

<div>4

<div>5</div>

</div>

</div>

</div>

</div>

<section id="log"></section>

 Run code snip

**Q=difference between @hostlistner and @hostbinding in angular 2?**

Ans=https://dzone.com/articles/what-are-hostbinding-and-hostlistener-in-angular

To understand @HostListener and @HostBinding, you should have basic knowledge about directives in Angular. There are three types of directives in Angular:

1. Component
2. Attribute Directive
3. Structural Directive

The basic difference between a component and a directive is that a component has a template, whereas an attribute or structural directive does not have a template. To understand these two concepts, let us start by creating a simple custom attribute directive. The directive below changes the background color of the host element:

import { Directive, ElementRef, Renderer } from '@angular/core';

@Directive({

selector: '[appChbgcolor]'

})

export class ChangeBgColorDirective {

constructor(private el: ElementRef, private renderer: Renderer) {

this.ChangeBgColor('red');

}

ChangeBgColor(color: string) {

this.renderer.setElementStyle(this.el.nativeElement, 'color', color);

}

}

To create a custom attribute directive, you need to create a class and decorate it with @Directive. In the constructor of the directive class, inject the objects **ElementRef** and **Renderer**. Instances of these two classes are needed to get the reference of the host element and of the renderer.

You can use the above attribute directive on a component template as shown in the code block below:

<div appChbgcolor>

<h3>{{title}}</h3>

</div>

Here, the component class holding the host element is created as below:

import { Component } from '@angular/core';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'Hey ng Developer ! ';

}

Right now, the **appChbgcolor** directive will change the color of the host element.

#### **@HostListener() Decorator**

In Angular, the @HostListener() function decorator allows you to handle events of the host element in the directive class.

Let's take the following requirement: when you hover you mouse over the host element, only the color of the host element should change. In addition, when the mouse is gone, the color of the host element should change to its default color. To do this, you need to handle events raised on the host element in the directive class. In Angular, you do this using **@HostListener()** .

To understand **@HostListener()** in a better way, consider another simple scenario: on the click of the host element, you want to show an alert window. To do this in the directive class, add @HostListener() and pass the event 'click' to it. Also, associate a function to raise an alert as shown in the listing below:

@HostListener('click') onClick() {

window.alert('Host Element Clicked');

}

In Angular, the @HostListener() function decorator makes it super easy to handle events raised in the host element inside the directive class. Let's go back to our requirement that says you must change the color to red only when the mouse is hovering, and when it's gone, the color of the host element should change to black. To do this, you need to handle the **mouseenter** and **mouseexit** events of the host element in the directive class. To achieve this, modify the **appChbgcolor** directive class as shown below:

import { Directive, ElementRef, Renderer, HostListener } from '@angular/core';

@Directive({

selector: '[appChbgcolor]'

})

export class ChangeBgColorDirective {

constructor(private el: ElementRef, private renderer: Renderer) {

// this.ChangeBgColor('red');

}

@HostListener('mouseover') onMouseOver() {

this.ChangeBgColor('red');

}

@HostListener('click') onClick() {

window.alert('Host Element Clicked');

}

@HostListener('mouseleave') onMouseLeave() {

this.ChangeBgColor('black');

}

ChangeBgColor(color: string) {

this.renderer.setElementStyle(this.el.nativeElement, 'color', color);

}

}

In the directive class, we are handling the **mouseenter** and **mouseexit** events. As you see, we are using @HostListener() to handle these host element events and assigning a function to it.

So, let's use **@HostListener()** function decorator to handle events of the host element in the directive class.

#### **@HostBinding() Decorator**

In Angular, the @HostBinding() function decorator allows you to set the properties of the host element from the directive class.

Let's say you want to change the style properties such as height, width, color, margin, border, etc., or any other internal properties of the host element in the directive class. Here, you'd need to use the @HostBinding() decorator function to access these properties on the host element and assign a value to it in directive class.

The @HostBinding() decorator takes one parameter, the name of the host element property which value we want to assign in the directive.

In our example, our host element is an HTML div element. If you want to set border properties of the host element, you can do that using @HostBinding() decorator as shown below:

@HostBinding('style.border') border: string;

@HostListener('mouseover') onMouseOver() {

this.border = '5px solid green';

}

Using this code, on a mouse hover, the host element border will be set to a green, solid 5-pixel width. Therefore, using the @HostBinding decorator, you can set the properties of the host element in the directive class.

Q=difference between event handling and event delegation?

Ans=

Q=