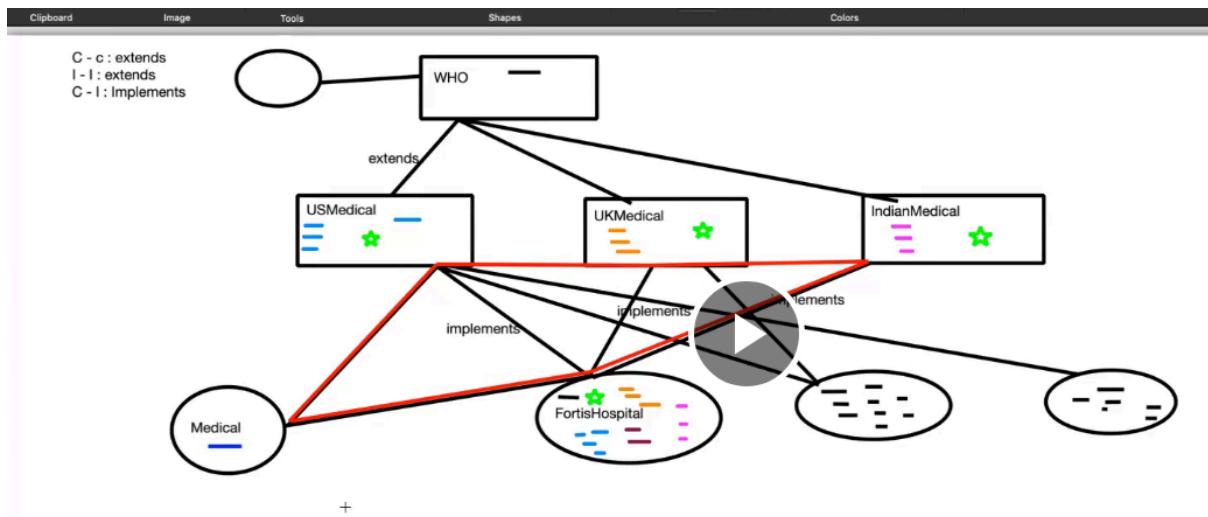


Multiple plus multi level inheritance-



Medical-

```

1 package InterfaceConcept;
2
3 public class Medical {
4
5     public void medicalNews() {
6         System.out.println("Medical -- news");
7     }
8
9
10
11
12 }
13

```

I

+

Play button

Fortis-

```

1 package InterfaceConcept;
2
3 public class FortisHospital extends Medical implements USMedical, UKMedical, IndianMedical {
4

```

Change sequence-

Error. — //Syntax error on token "extends", permits expected

```

1 package InterfaceConcept;
2
3 public class FortisHospital implements USMedical, UKMedical, IndianMedical extends Medical {
4

```

Test-

```

35     fh.medicalNews();

```

Medical news.

Medical-

One more method.

```

8
9  public void medicalRD() {
10    System.out.println("medical R&D");
11 }
12

```

Override in fortis-

```

110
111=   @Override
112  public void medicalRD() {
113    System.out.println("FH R&D");
114  }
115

```

Test-

```

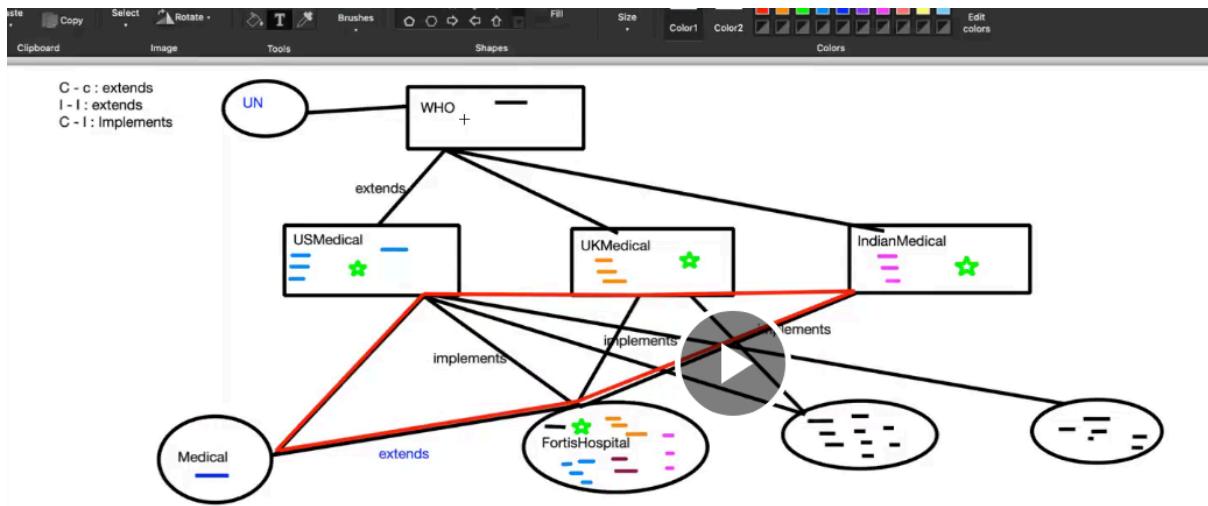
36     fh.medicalRD();

```

Fortis hospital medicalrd is called.

Interface parent is class-

Is it possible?



Un-

```

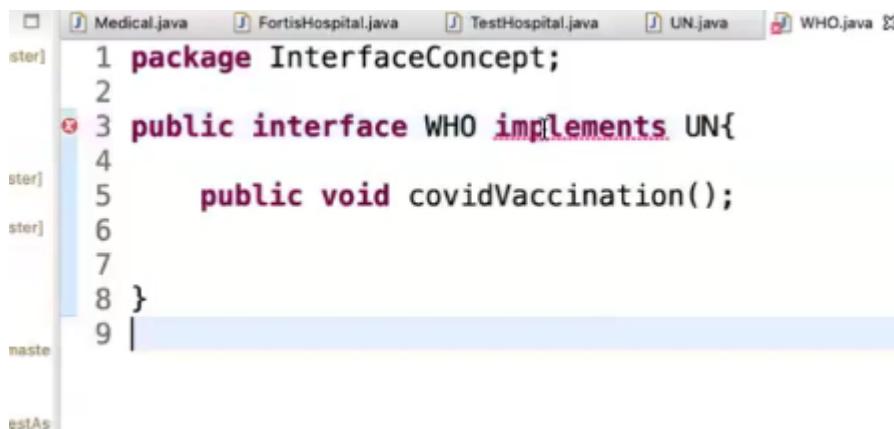
1 package InterfaceConcept;
2
3 public class UN {
4
5     public void medicalFunds() {
6         System.out.println("UN -- medical funds");
7     }
8
9
10}
11
12

```

Who-

Cant implement class.

error - Syntax error on token "implements", permits expected



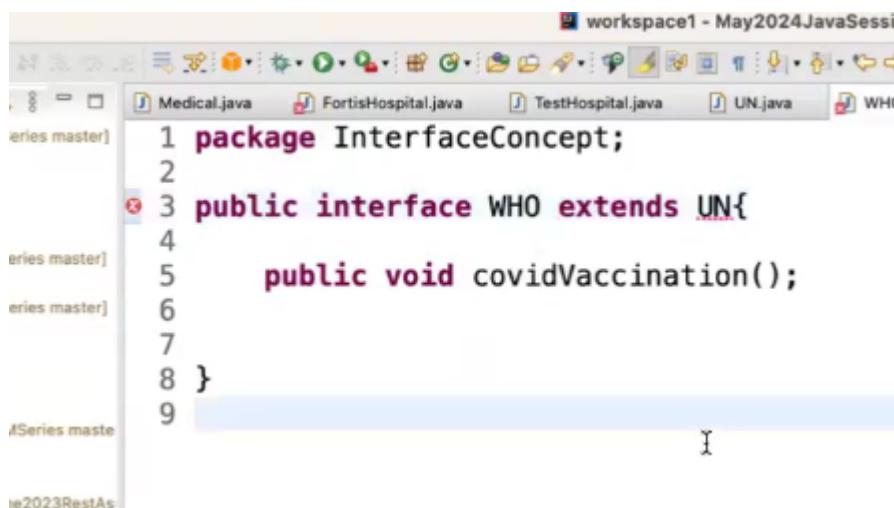
```

1 package InterfaceConcept;
2
3 public interface WHO implements UN{
4
5     public void covidVaccination();
6
7
8 }
9
  
```

Extends-

Error - //The type un4 cannot be a superinterface of who4; a superinterface must be an interface

Cant do



```

1 package InterfaceConcept;
2
3 public interface WHO extends UN{
4
5     public void covidVaccination();
6
7
8 }
9
  
```

Note-

Interface cannot have parent which is a normal class.

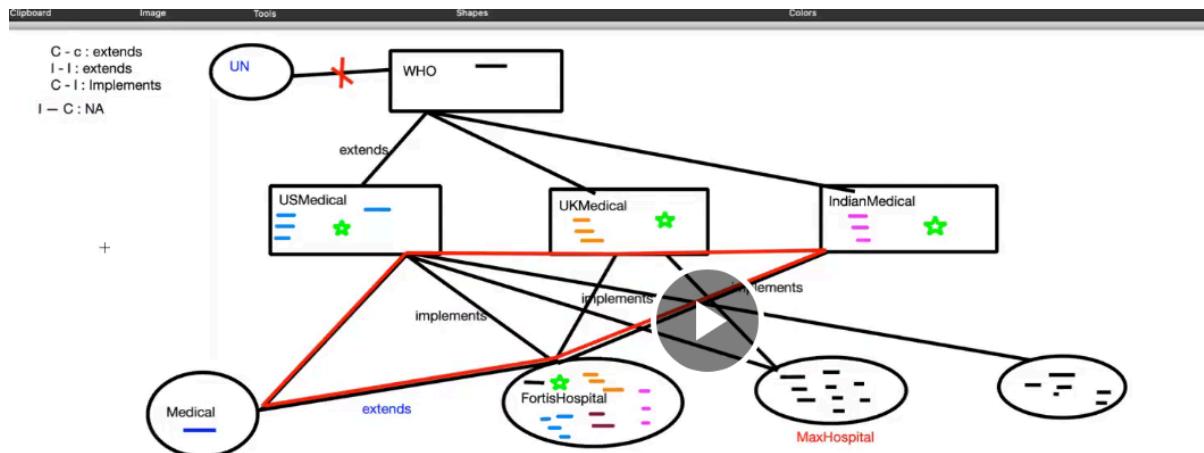
Interface can have another interface as parent.

Composition-

Has a relationship.

Sibling added.

Max and fortis.



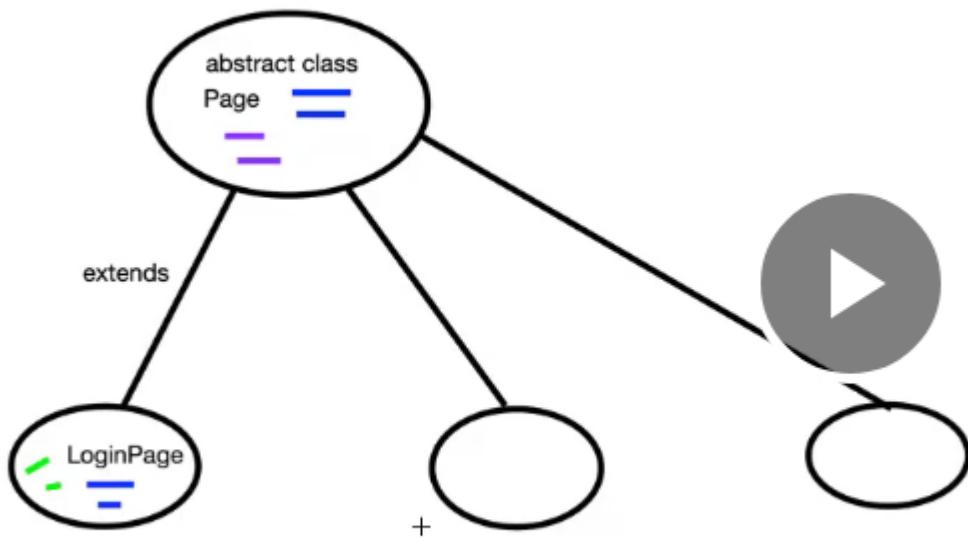
```

5 //interface vars are static and final in nature by default
6
7 //only method declaration: no method body
8 //only method prototype
9 //abstract methods: a method without body(only for non static methods)
10 //can not create the object of interface
11 //can not create the const.. of the Interface
12 //static method with body
13
14
15
16

```

As no object can be created of interface so no constructor.

Abstract versus interface-



In interface not mandatory to write abstract word for methods-

us medical interface-

```
14  
15  
16  
17     public abstract void physioServices();  
18
```

paste page5, login5, testfor5-

The screenshot shows a Java code editor with the file `page5.java` open. The code defines an abstract class `page5` with various methods, some of which are static and final. The code uses `System.out.println` statements to output messages to the console.

```
1 package com.day20;
2
3 public abstract class page5 {
4
5     //class with abstract and non abstract methods.
6
7     // Abstract methods
8     public abstract void title();           // without return
9     public abstract String url();          // with return
10
11    // Non-abstract method without return
12    public void loading() {
13        System.out.println("Page is loading...");
14    }
15
16    // Non-abstract method with return
17    public boolean unloading() {
18        System.out.println("Page is unloading...");
19        return true;
20    }
21
22    //create static method without return.
23    public static void test() {
24        System.out.println("test static without return method");
25    }
26
27    public static void test1() {
28        System.out.println("test static without return method");
29    }
30
31    //create static method with return.
32    public static String getstatic(String name) {
33        System.out.println("static method with return" + name);
34        return "tiger";
35    }
36
37    //create final method without return.
38    public final void getfinal(String name) {
39        System.out.println("final method without return" + name);
40    }
41
42    //create final method with return.
43    public final String getfinal1(String name) {
44        System.out.println("final method with return" + name);
45        return "lion";
46    }
47
48    //create static final method without return.
49    public static final void getstaticfinal(String name) {
50        System.out.println("static final method without return" + name);
51    }
52}
```

```

46     System.out.println("static final method without return" + name);
47 }
48
49 //create static final method with return.
50@ public static final String getstaticfinal1(String name) {
51     System.out.println("static final method with return" + name);
52     return "tiger";
53 }
54
55 //create default method in abstract with return.
56 //Default methods are allowed only in interfaces.
57// default String getdefault(String name) {
58//     System.out.println("default method with return" + name);
59//     return "tiger";
60// }
61//
62// //create default method in abstract without return.
63// Default methods are allowed only in interfaces.
64// default void getdefault1(String name) {
65//     System.out.println("default method without return" + name);
66// }
67 }
68

```

```

1 package com.day20;
2
3 public class login5 extends page5 {
4
5     @Override
6@     public void title() {
7         System.out.println("child class login title overridden method");
8     }
9
10    @Override
11@    public String url() {
12        System.out.println("child class url overridden method");
13        return "gorilla";
14    }
15
16    @Override
17@    public void loading() {
18        System.out.println("child Page is loading...");
19    }
20
21    //override not with static.
22    //The method getstatic(String) of type login5 must override
23    //or implement a supertype method
24// @Override
25// public static String getstatic(String name) {
26//     System.out.println("static method with return" + name);
27//     return "tiger";
28// }
29

```

```
26 //      System.out.println(" static method with return " + name);
27 //      return "tiger";
28 //  }
29
30 public static String getstatic(String name) {
31     System.out.println("static method with return from child" + name);
32     return "chimpanzee";
33 }
34
35 // @Override
36 // //cannot use override with final.
37 // //Cannot override the final method from page5
38 // public final void getfinal(String name) {
39 //     System.out.println("final method without return" + name);
40 // }
41
42 ////cannot use without annotation also.
43 ////Cannot override the final method from page5
44 // public final void getfinal(String name) {
45 //     System.out.println("final method without return" + name);
46 // }
47
48 // @Override
49 // //cannot use override.
50 // //The method getstaticfinal1(String) of type login5 must override
51 // //or implement a supertype method
52 // public static final String getstaticfinal1(String name) {
53 //     System.out.println("static final method with return" + name);
54 //     return "tiger";
55 // }
56
57 //without override also final not allowed.
58 ////Cannot override the final method from page5
59 // public static final String getstaticfinal1(String name) {
60 //     System.out.println("static final method with return" + name);
61 //     return "tiger";
62 // }
63
64 }
65
```

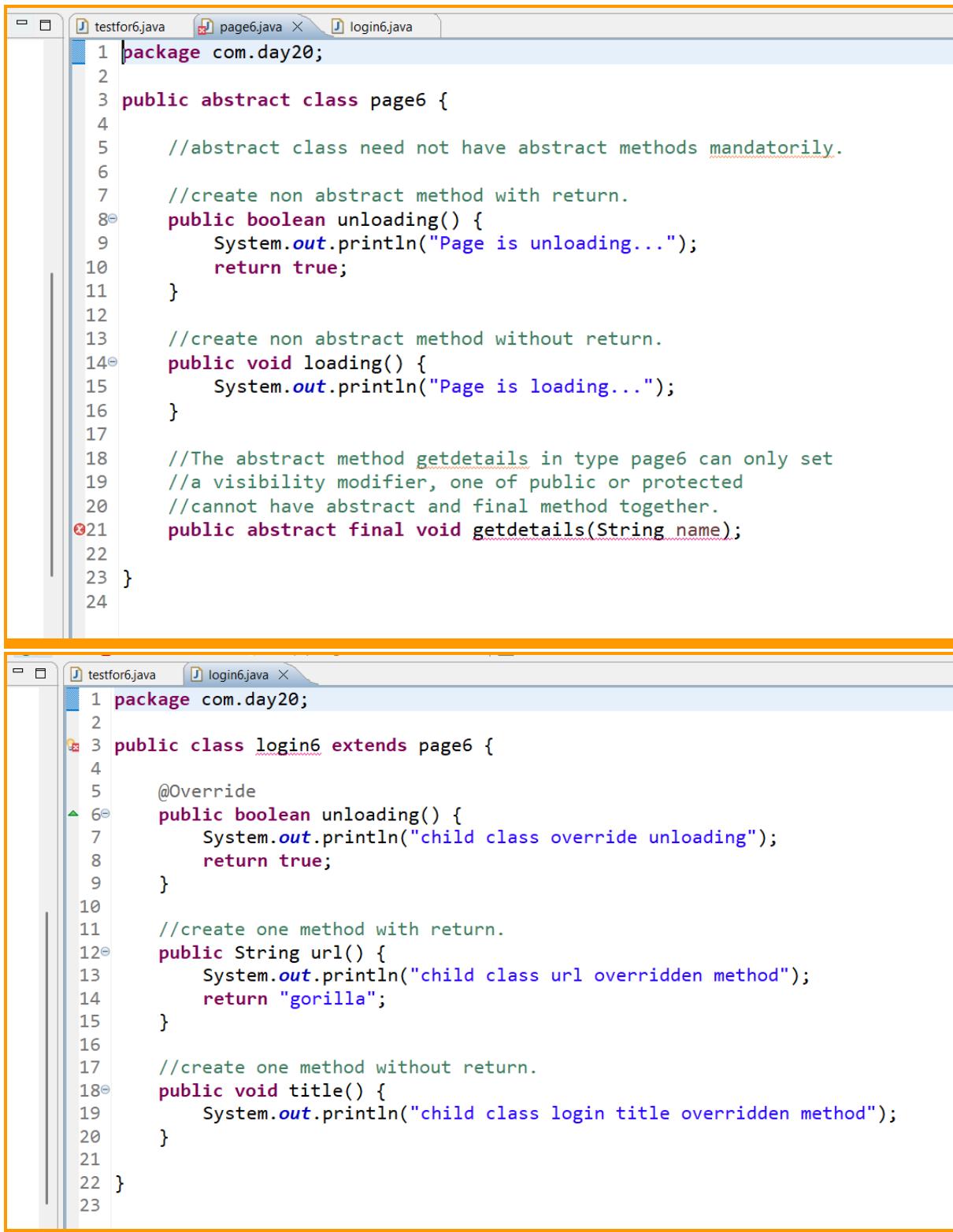
```

testfor5.java X
1 package com.day20;
2
3 public class testfor5 {
4
5     public static void main(String[] args) {
6
7         login5 l1 =new login5();
8         l1.title();
9         String s1=l1.url();
10        System.out.println(s1);
11        l1.loading();
12        String s2    =l1.getstatic("karan");
13        //warning for static.
14        //The static method getstatic(String) from the type
15        //login5 should be accessed in a static way
16        System.out.println(s2);
17        String s3=login5.getstatic("australia");
18        System.out.println(s3);
19        l1.getfinal("georgia");
20        String s4=l1.getfinal1("cambodia");
21        System.out.println(s4);
22        l1.loading();
23        boolean b1=l1.unloading();
24        System.out.println(b1);
25        l1.getstaticfinal("gimper");
26        //warning for static.
27        //The static method getstaticfinal(String) from the type page5
28        //should be accessed in a static way
29
30        //warning for static.
31        //The static method getstaticfinal(String) from the type page5
32        //should be accessed in a static way
33        login5.getstaticfinal("usa");
34        String s5=l1.getstaticfinal1("japan");
35        System.out.println(s5);
36        //static warning.
37        //The static method getstaticfinal1(String) from
38        //the type page5 should be accessed in a static way
39        String s6=login5.getstaticfinal1("canada");
40        System.out.println(s6);
41        l1.test();
42
43        //cannot create object of abstract.
44        //Cannot instantiate the type page6
45        //page6 p1=new page6();
46
47        //can access only page methods not the child classes.
48        System.out.println("-----");
49        page6 p1=new login6();
50        p1.loading();
51        boolean b2=p1.unloading();
52        System.out.println(b2);

```

```
46     p1.loading(),
47     boolean b2=p1.unloading();
48     System.out.println(b2);
49
50
51     }
52 }
53
54 }
55
56 //child class login title overridden method
57 //child class url overridden method
58 //gorilla
59 //child Page is loading...
60 //static method with return from childkaran
61 //chimpanzee
62 //static method with return from childaustralia
63 //chimpanzee
64 //final method without returngeorgia
65 //final method with returncambodia
66 //lion
67 //child Page is loading...
68 //Page is unloading...
69
70 //child Page is loading...
71 //Page is unloading...
72 //true
73 //static final method without returngimper
74 //static final method without returnusa
75 //static final method with returnjapan
76 //tiger
77 //static final method with returncanada
78 //tiger
79 //test static without return method
80 //-----
81 //Page is loading...
82 //child class override unloading
83 //true
84
85
86
```

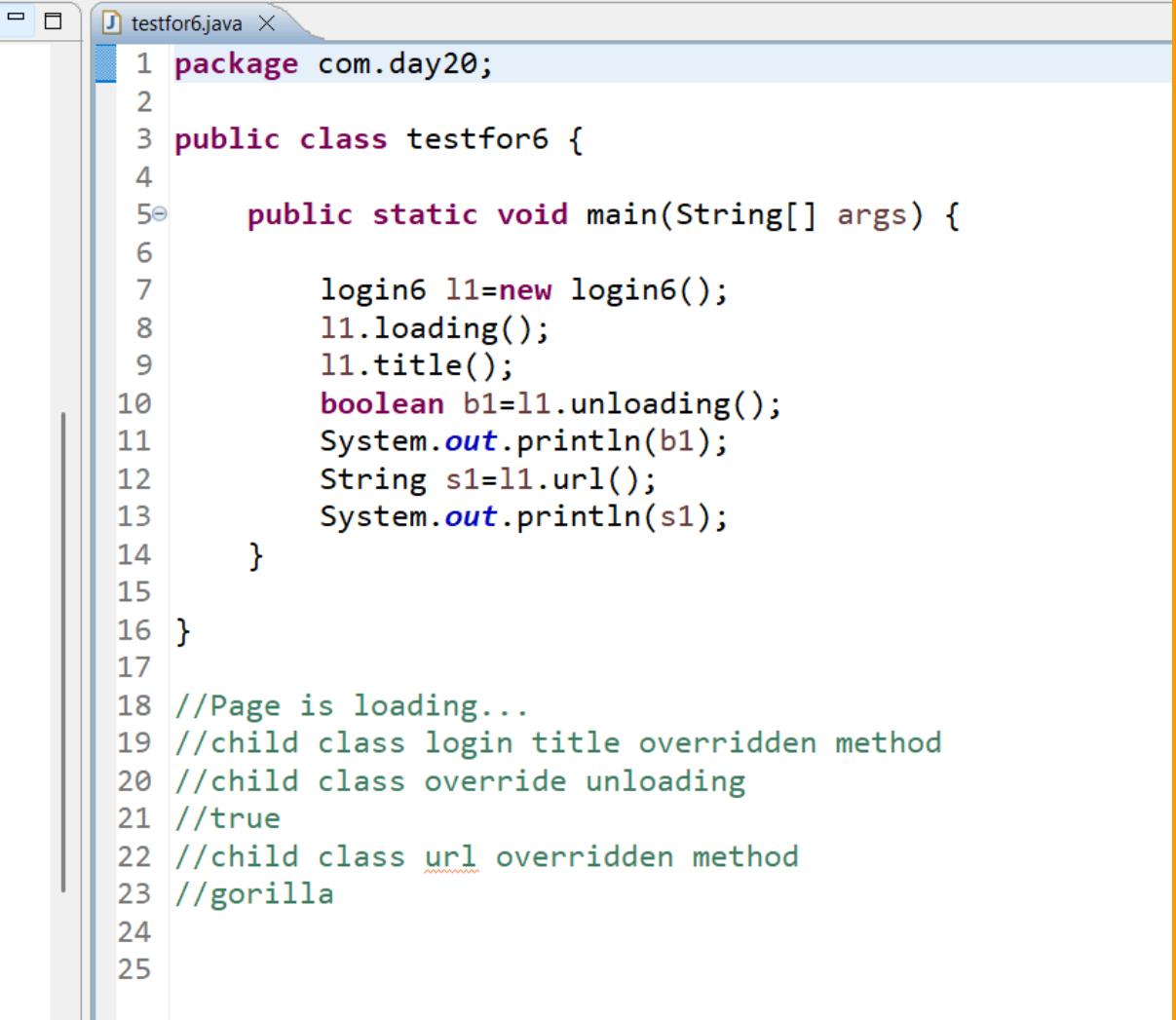
paste page6, login6, testfor6-



```
testfor6.java page6.java login6.java
1 package com.day20;
2
3 public abstract class page6 {
4
5     //abstract class need not have abstract methods mandatorily.
6
7     //create non abstract method with return.
8     public boolean unloading() {
9         System.out.println("Page is unloading...");
10        return true;
11    }
12
13    //create non abstract method without return.
14    public void loading() {
15        System.out.println("Page is loading...");
16    }
17
18    //The abstract method getdetails in type page6 can only set
19    //a visibility modifier, one of public or protected
20    //cannot have abstract and final method together.
21    public abstract final void getdetails(String name);
22
23 }
24
```



```
testfor6.java login6.java
1 package com.day20;
2
3 public class login6 extends page6 {
4
5     @Override
6     public boolean unloading() {
7         System.out.println("child class override unloading");
8         return true;
9     }
10
11    //create one method with return.
12    public String url() {
13        System.out.println("child class url overridden method");
14        return "gorilla";
15    }
16
17    //create one method without return.
18    public void title() {
19        System.out.println("child class login title overridden method");
20    }
21
22 }
23
```



```
1 package com.day20;
2
3 public class testfor6 {
4
5     public static void main(String[] args) {
6
7         login6 l1=new login6();
8         l1.loading();
9         l1.title();
10        boolean b1=l1.unloading();
11        System.out.println(b1);
12        String s1=l1.url();
13        System.out.println(s1);
14    }
15
16 }
17
18 //Page is loading...
19 //child class login title overridden method
20 //child class override unloading
21 //true
22 //child class url overridden method
23 //gorilla
24
25
```

Abstract class-

Can have abstract and non abstract methods.

Not mandatory to write abstract methods. it means if the class is declared as abstract, but there are no abstract methods, it wont throw error.



```

1 package Abstraction;
2
3 public abstract class Page {
4
5     //we can have both abstract and non abstract methods
6
7
8     //abstract methods
9 //    public abstract void title();
10 //    public abstract void url();
11
12
13     //non abstract methods
14     public void loading() {
15         System.out.println("page loading");
16     }
17
18
19
20
21 }
22

```

```

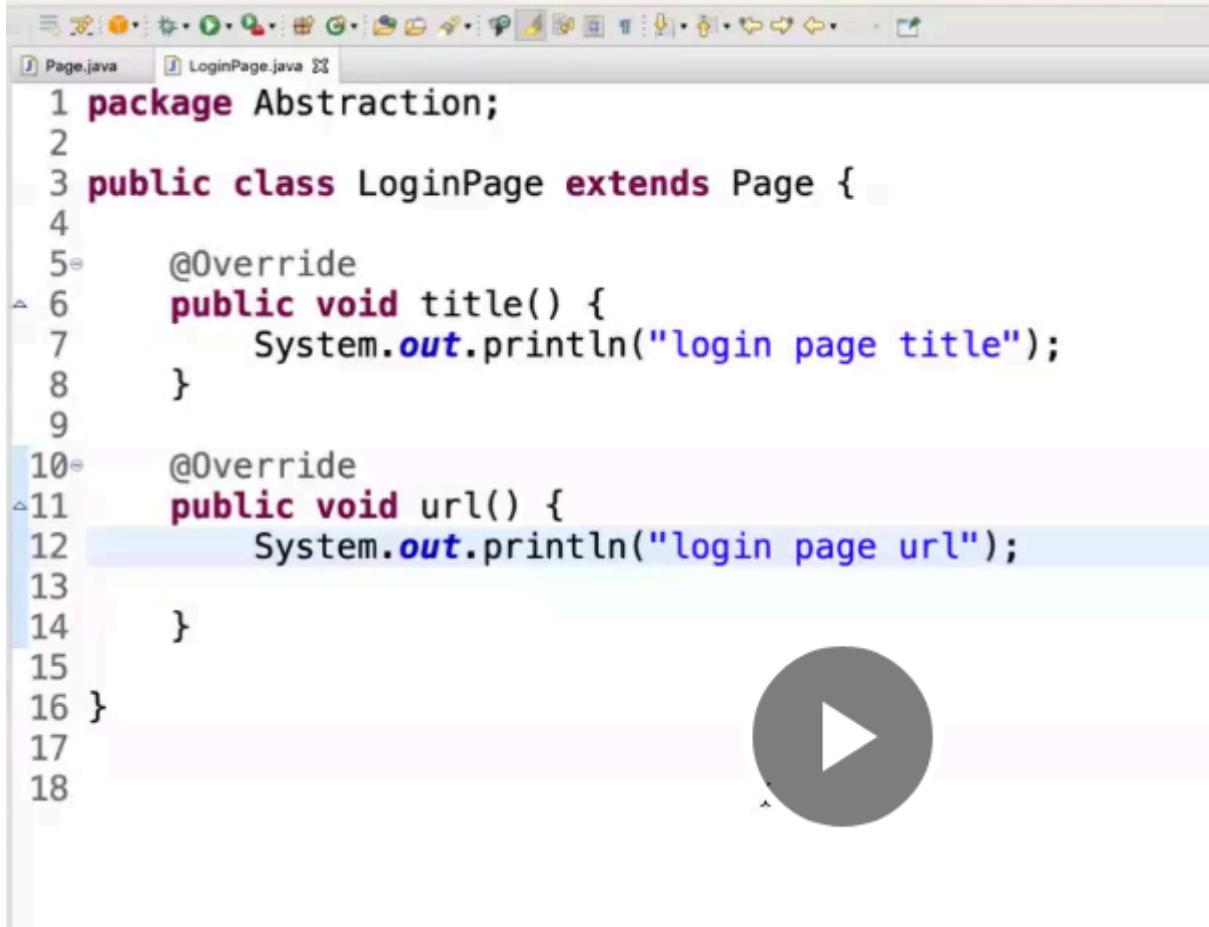
3 public abstract class Page {
4
5     //we can have both abstract and non abstract methods
6     //0% abstraction -- yes
7     //100% abstraction -- yes
8     //partial abstraction -- yes
9
10
11     //abstract methods
12     public abstract void title();
13     public abstract void url();
14

```

Try to create static methods in abstract class – hw.
- allowed/

Try to create default methods in abstract class –
hw. - not allowed. — Default methods are allowed only in interfaces.

Login page-



```

1 package Abstraction;
2
3 public class LoginPage extends Page {
4
5     @Override
6     public void title() {
7         System.out.println("login page title");
8     }
9
10    @Override
11    public void url() {
12        System.out.println("login page url");
13    }
14
15
16 }
17
18

```

Abstract methods are mandatory to implement.

Abstract class-



```

15
16     //non abstract methods
17     public void loading() {
18         System.out.println("page loading in 20 secs");
19     }
20

```

If you want to override-

Login page

```

15
16  @Override
17  public void loading() {
18      System.out.println("Login page Loading in 5 secs");
19  }
20
21

```



Final methods cannot be overridden-
abstract class-

```

15
16  //non abstract methods
17  public final void loading() {
18      System.out.println("page loading in 20 secs");
19  }
20

```

Abstract class-
Final method

```

21
22  public final void displayLogo() {
23      System.out.println("mylogo.jpg");
24  }
25

```

Login page-

```

21
22  public final void displayLogo() {
23      System.out.println("mylogo.jpg");
24  }
25 }
26

```

Error when overriding final - *//Cannot override the final method from page5*

Login-
Individual method

```
21 //individual  
22 public void doLogin() {  
23     System.out.println("login to app");  
24 }
```

Test-

The screenshot shows an IDE interface with three tabs at the top: Page.java, LoginPage.java, and TestPage.java. The TestPage.java tab is active, displaying the following code:

```
1 package Abstraction;  
2  
3 public class TestPage {  
4  
5     public static void main(String[] args) {  
6  
7         LoginPage lp = new LoginPage();  
8         lp.title();  
9         lp.url();  
10        lp.loading();  
11        lp.doLogin();  
12  
13  
14    }  
15  
16}  
17 }
```

Below the code editor, the terminal window shows the execution of the TestPage class. The output is:

```
<terminated> TestPage {1} [Java Application] /Users/naveenautomationlabs/p2/pool  
login page title  
login page url  
Login page loading in 5 secs  
login to app
```

```

4
5 //we can have both abstract and non abstract methods
6 //0% abstraction -- yes
7 //100% abstraction -- yes
8 //partial abstraction -- yes
9
10 //can not create the object of abstract class
11

13
14 //can not create the object of abstract class
15 // Page p = new Page();
16

```

Top cast-

Cannot access the individual method of child.

```

17 //top casting:
18 //child class object can be referred by abstract parent class ref variable
19 Page pg = new LoginPage();
20 pg.title();
21 pg.url();
22 pg.loading();
23
24
25 }

login page title
login page url
Login page loading in 5 secs

```

Downcast-

Not allowed

paste page7, login7, testfor7-

The screenshot shows a Java development environment with two code editors. The top editor contains the `page7.java` file, and the bottom editor contains the `login7.java` file. Both files are part of the package `com.day20`.

```
page7.java content:
1 package com.day20;
2
3 public abstract class page7 {
4
5     //create one abstract method with return.
6     public abstract boolean unloading();
7
8     //create one abstract method without return.
9     public abstract void loading();
10
11    //create normal method with return.
12    public String url() {
13        System.out.println("parent class url method");
14        return "gorilla";
15    }
16
17    //create normal method without return.
18    public void title() {
19        System.out.println("parent class login title method");
20    }
21
22 }
23
```

```
login7.java content:
1 package com.day20;
2
3 public class login7 extends page7 {
4
5     @Override
6     public boolean unloading() {
7         System.out.println("child class override unloading");
8         return false;
9     }
10
11     @Override
12     public void loading() {
13         System.out.println("child class override loading");
14     }
15
16
17 }
18
```

```
testfor7.java
1 package com.day20;
2
3 public class testfor7 {
4
5     public static void main(String[] args) {
6
7         login7 l1=new login7();
8
9         l1.loading();
10        l1.title();
11        boolean b1=l1.unloading();
12        System.out.println(b1);
13        String s1=l1.url();
14        System.out.println(s1);
15
16        //downcast not allowed.
17        //Cannot instantiate the type page7
18        page7 p1=(login7)new page7();
19
20    }
21
22 }
23
24
25 //child class override loading
26 //parent class login title method
27 //child class override unloading
28 //false
29 //parent class url method
30 //gorilla
31
32
```

Constructor allowed in abstract class-



```

1 package Abstraction;
2
3 public abstract class Page {
4
5     //we can have both abstract and non abstract methods
6     //0% abstraction -- yes
7     //100% abstraction -- yes
8     //partial abstraction -- yes
9
10    //can not create the object of abstract class
11
12    //but can we have const... of the abstract class: this is allowed
13    //and it will be called when you create the object of the child class
14
15
16    public Page() {
17        System.out.println("page const...default");
18    }
19
20
21    //abstract methods

```

test page-

```

1 package Abstraction;
2
3 public class TestPage {
4
5     public static void main(String[] args) {
6
7         LoginPage lp = new LoginPage();
8         lp.title();
9         lp.url();
10        lp.loading();
11        lp.doLogin();
12

```

<terminated> TestPage (1) [Java Application] /Users/naveenautomationlab/

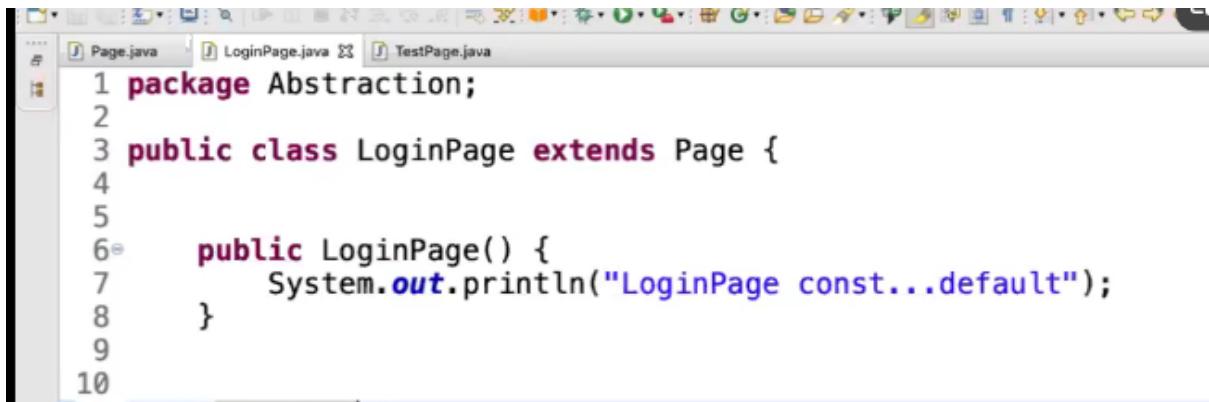
```

page const...default
login page title
login page url
Login page loading in 5 secs
login to app

```

Login page-

Same constructor as abstract class.

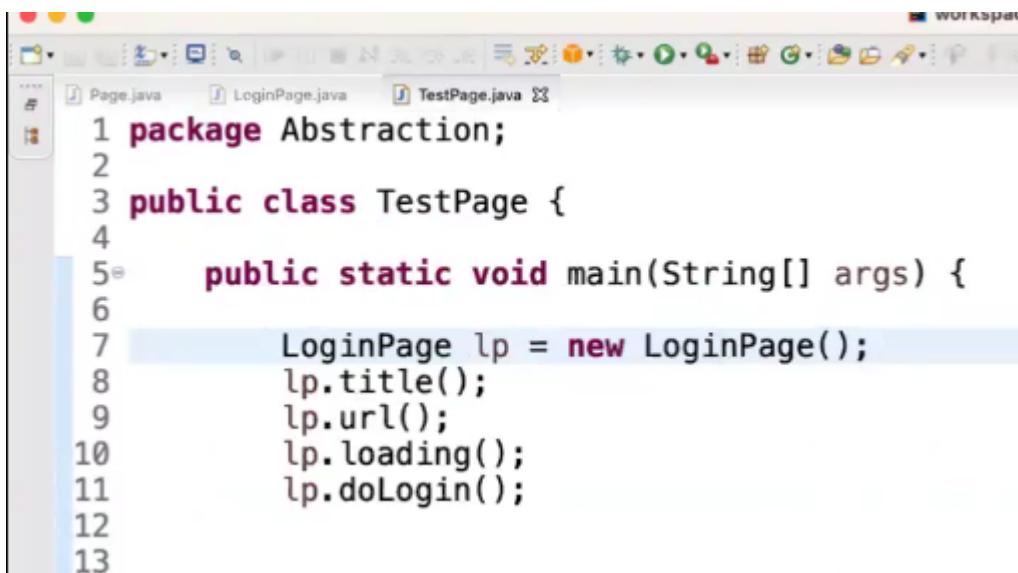


```

1 package Abstraction;
2
3 public class LoginPage extends Page {
4
5
6    public LoginPage() {
7        System.out.println("LoginPage const...default");
8    }
9
10

```

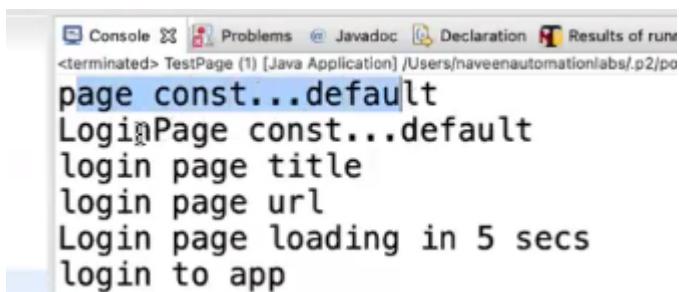
Test-



```

1 package Abstraction;
2
3 public class TestPage {
4
5    public static void main(String[] args) {
6
7        LoginPage lp = new LoginPage();
8        lp.title();
9        lp.url();
10       lp.loading();
11       lp.doLogin();
12
13

```



```

<terminated> TestPage (1) [Java Application] /Users/naveenautomation/labs/p2/po
page const...default
LoginPage const...default
login page title
login page url
Login page loading in 5 secs
login to app

```

First parent class constructor called then child class constructor called.

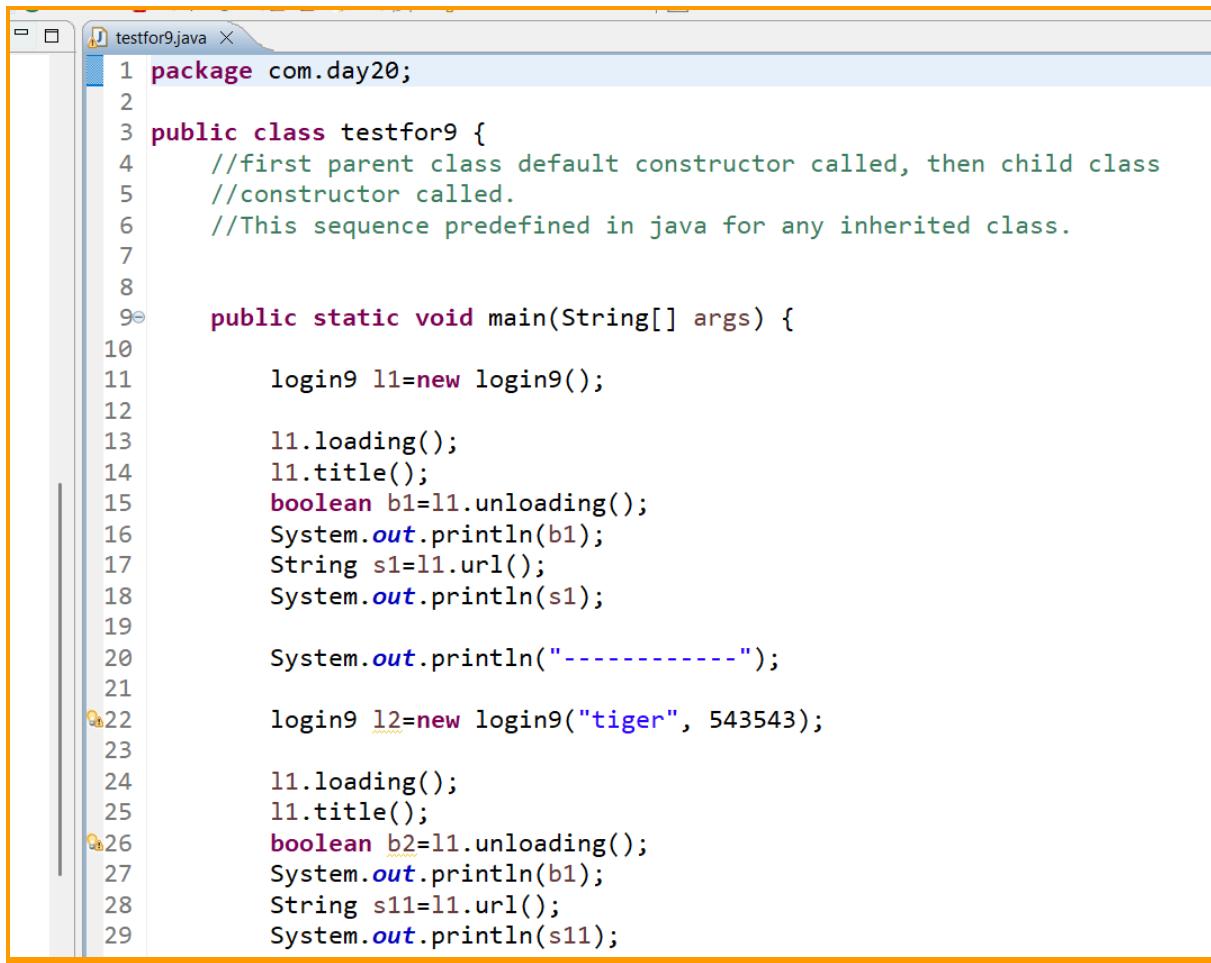
This sequence predefined in java for any inherited class.

paste page9, login9, testfor9-

The screenshot shows a Java development environment with three tabs at the top: "login9.java", "page9.java", and "testfor9.java". The "page9.java" tab is active, indicating it is the current file being edited.

```
1 package com.day20;
2
3 public abstract class page9 {
4
5     String name;
6     int id;
7
8     //constructor allowed in abstract class.
9     public page9() {
10         System.out.println("default constructor");
11     }
12
13     public page9(String name, int id) {
14         this.name=name;
15         this.id=id;
16     }
17
18
19     //create one abstract method with return.
20     public abstract boolean unloading();
21
22     //create one abstract method without return.
23     public abstract void loading();
24
25     //create normal method with return.
26     public String url() {
27         System.out.println("parent class url method");
28         return "gorilla";
29     }
30
31     //create normal method without return.
32     public void title() {
33         System.out.println("parent class login title method");
34     }
35
36 }
37
```

```
1 package com.day20;
2
3 public class login9 extends page9 {
4
5     //name and id taken from parent itself.
6
7     //constructor in child class.
8     public login9() {
9         System.out.println("child class default constructor");
10    }
11
12     public login9(String name, int id) {
13         this.name=name;
14         this.id=id;
15         System.out.println("child class parameter constructor.");
16         System.out.println(this.name);
17         System.out.println(this.id);
18    }
19
20    @Override
21    public boolean unloading() {
22        System.out.println("child class override unloading");
23        return false;
24    }
25
26    @Override
27    public void loading() {
28        System.out.println("child class override loading");
29    }
30
31
32 }
33 }
```



```
1 package com.day20;
2
3 public class testfor9 {
4     //first parent class default constructor called, then child class
5     //constructor called.
6     //This sequence predefined in java for any inherited class.
7
8
9     public static void main(String[] args) {
10
11         login9 l1=new login9();
12
13         l1.loading();
14         l1.title();
15         boolean b1=l1.unloading();
16         System.out.println(b1);
17         String s1=l1.url();
18         System.out.println(s1);
19
20         System.out.println("-----");
21
22         login9 l2=new login9("tiger", 543543);
23
24         l2.loading();
25         l2.title();
26         boolean b2=l2.unloading();
27         System.out.println(b1);
28         String s11=l2.url();
29         System.out.println(s11);
```

```
28         String s11=l1.url();
29         System.out.println(s11);
30
31     }
32
33 }
34
35
36 //default constructor
37 //child class default constructor
38 //child class override loading
39 //parent class login title method
40 //child class override unloading
41 //false
42 //parent class url method
43 //gorilla
44 -----
45 //default constructor
46 //child class parameter constructor.
47 //tiger
48 //543543
49 //child class override loading
50 //parent class login title method
51 //child class override unloading
52 //false
53 //parent class url method
54 //gorilla
```

```
-----  
53 //parent class url method  
54 //gorilla  
55  
56  
57  
58  
59
```

paste page10, login10, testfor10-

```
1 package com.day20;
2
3 public abstract class page10 {
4
5     String name;
6     int id;
7
8     //only parameter constructor.
9     public page10(String name, int id) {
10         this.name=name;
11         this.id=id;
12         System.out.println(this.name);
13         System.out.println(this.id);
14         System.out.println("parent class param constructor.");
15     }
16
17
18     //create one abstract method with return.
19     public abstract boolean unloading();
20
21     //create one abstract method without return.
22     public abstract void loading();
23
24     //create normal method with return.
25     public String url() {
26         System.out.println("parent class url method");
27         return "gorilla";
28     }
29
30     return "gorilla";
31 }
32
33 //create normal method without return.
34 public void title() {
35     System.out.println("parent class login title method");
36 }
```

```
1 package com.day20;
2
3 public class login10 extends page10 {
4
5     //name and id taken from parent itself.
6
7     //constructor in child class.
8     //Implicit super constructor page10() is undefined.
9     //Must explicitly invoke another constructor
10    //when parent doesn't have default constructor then child also can't have.
11
12    //    public login10() {
13    //        //if we add super it says there is no default page10 constructor.
14    //        super(); //The constructor page10() is undefined
15    //        System.out.println("child class default constructor");
16    //    }
17
18    //have to call one of the constructor from parent to work.
19    public login10() {
20        super("karan", 3445);
21        System.out.println("default constructor of child class.");
22    }
23
24    //Implicit super constructor page10() is undefined.
25    //Must explicitly invoke another constructor
26    //    public login10(String name, int id) {
27    //        this.name=name;
28    //        this.id=id;
29    //        System.out.println("child class parameter constructor.");
30    //        System.out.println(this.name);
31    //        System.out.println(this.id);
32    //    }
33
34    //have to call one of the constructor from parent to work.
35    public login10(String name, int id) {
36        super(name, id);
37        this.name=name;
38        this.id=id;
39        System.out.println("child class parameter constructor.");
40        System.out.println(this.name);
41        System.out.println(this.id);
42    }
43}
```

```

41         System.out.println(this.id);
42     }
43
44     @Override
45     public boolean unloading() {
46         System.out.println("child class override unloading");
47         return false;
48     }
49
50     @Override
51     public void loading() {
52         System.out.println("child class override loading");
53     }
54
55
56 }
57

```

testfor10.java

```

1 package com.day20;
2
3 public class testfor10 {
4     //first parent class default constructor called, then child class
5     //constructor called.
6     //This sequence predefined in java for any inherited class.
7
8
9     public static void main(String[] args) {
10
11         login10 l1=new login10();
12
13         l1.loading();
14         l1.title();
15         boolean b1=l1.unloading();
16         System.out.println(b1);
17         String s1=l1.url();
18         System.out.println(s1);
19
20         System.out.println("-----");
21
22         login10 l2=new login10("tiger", 543543);
23
24         l2.loading();
25         l2.title();
26         boolean b2=l2.unloading();
27         System.out.println(b2);
28         String s11=l2.url();
29         System.out.println(s11);
30

```

```
27         System.out.println(s1),
28     String s11=l1.url();
29     System.out.println(s11);
30
31 }
32
33 }
34
35
36 //karan
37 //3445
38 //parent class param constructor.
39 //default constructor of child class.
40 //child class override loading
41 //parent class login title method
42 //child class override unloading
43 //false
44 //parent class url method
45 //gorilla
46 //-----
47 //tiger
48 //543543
49 //parent class param constructor.
50 //child class parameter constructor.
51 //tiger
```

```
50 //child class parameter constructor.  
51 //tiger  
52 //543543  
53 //child class override loading  
54 //parent class login title method  
55 //child class override unloading  
56 //false  
57 //parent class url method  
58 //gorilla  
59  
60  
61  
62  
63  
64
```

paste page11-

The screenshot shows a Java code editor with the file `page11.java` open. The code defines an abstract class `page11` with several methods and a constructor. The code is color-coded for syntax: package names in purple, class and interface names in blue, and keywords in red. Error markers (red exclamation marks) are placed on lines 7, 9, 11, and 12, indicating undeclared variables `this.name`, `this.id`, and `this.name` respectively.

```
1 package com.day20;
2
3 public abstract class page11 {
4
5     //only parameter constructor.
6     public page11(String name, int id) {
7         this.name=name; //when variable not declared then error.
8         //name cannot be resolved or is not a field
9         this.id=id;
10        //id cannot be resolved or is not a field
11        System.out.println(this.name);
12        System.out.println(this.id);
13        System.out.println("parent class param constructor.");
14    }
15
16
17    //create one abstract method with return.
18    public abstract boolean unloading();
19
20    //create one abstract method without return.
21    public abstract void loading();
22
23    //create normal method with return.
24    public String url() {
25        System.out.println("parent class url method");
26        return "gorilla";
27    }
28
29    //create normal method without return.
30    public void title() {
31        System.out.println("parent class login title method");
32    }
33
34 }
```

Parent class no constructor-



```

1 package Abstraction;
2
3 public abstract class Page {
4
5     //we can have both abstract and non abstract methods
6     //0% abstraction -- yes
7     //100% abstraction -- yes
8     //partial abstraction -- yes
9
10    //can not create the object of abstract class
11
12    //but can we have const... of the abstract class: this is allowed
13    //and it will be called when you create the object of the child class
14
15
16 //    public Page() {
17 //        System.out.println("page const...default");
18 //    }
19
20
21 //abstract methods
22 public abstract void title();

```

Login page with constructor-

```

1 package Abstraction;
2
3 public class LoginPage extends Page {
4
5
6     public LoginPage() {
7         System.out.println("LoginPage const...default");
8     }
9
10
11     @Override

```

Test-

The screenshot shows the Eclipse IDE interface. In the top-left corner, there are three tabs: 'Page.java', 'LoginPage.java', and 'TestPage.java'. The 'TestPage.java' tab is currently active, displaying the following Java code:

```
1 package Abstraction;
2
3 public class TestPage {
4
5     public static void main(String[] args) {
6
7         LoginPage lp = new LoginPage();
8         lp.title();
9         lp.url();
10        lp.loading();
11        lp.doLogin();
12
13
14     // Don't create the object of abstract class
15 }
```

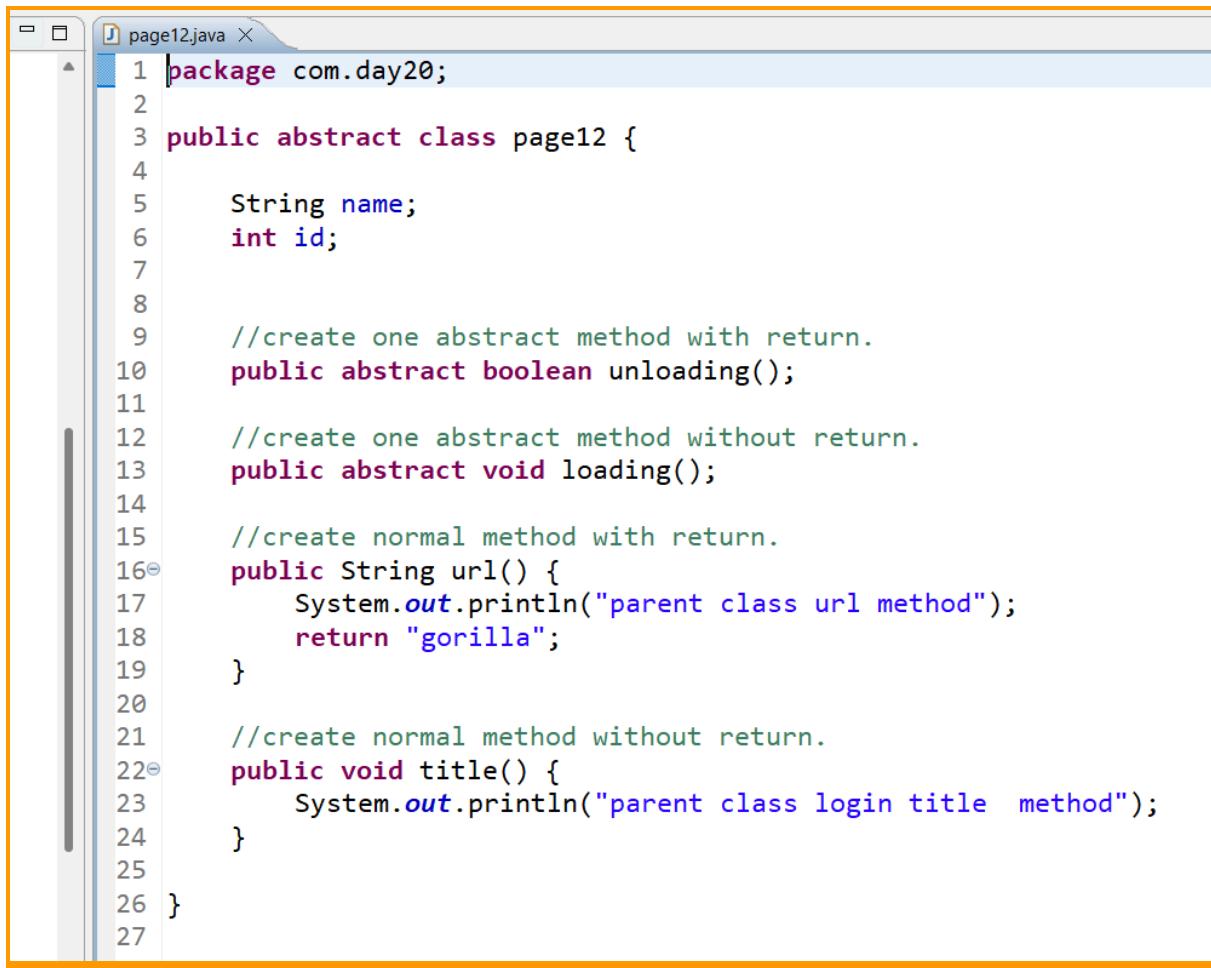
In the bottom-right corner of the code editor, there is a note: 'Don't create the object of abstract class'.

Below the code editor is the Eclipse IDE's toolbar with various icons.

The bottom half of the screenshot shows the 'Console' view, which displays the following output:

```
24JavaSession | Abstraction/TestPage.java - Eclipse IDE
[Run] [Stop] [Minimize] [Maximize]
Console Problems Javadoc Declaration Results of run
<terminated> TestPage (1) [Java Application] /Users/naveenautomationlabs/p2/
LoginPage const...default
login page title
login page url
Login page loading in 5 secs
login to app
```

paste page12, login12, testfor12-



The screenshot shows a Java code editor window with the file "page12.java" open. The code defines an abstract class "page12" with various methods and fields. The code is color-coded for syntax highlighting.

```
1 package com.day20;
2
3 public abstract class page12 {
4
5     String name;
6     int id;
7
8
9     //create one abstract method with return.
10    public abstract boolean unloading();
11
12    //create one abstract method without return.
13    public abstract void loading();
14
15    //create normal method with return.
16    public String url() {
17        System.out.println("parent class url method");
18        return "gorilla";
19    }
20
21    //create normal method without return.
22    public void title() {
23        System.out.println("parent class login title method");
24    }
25
26 }
27
```

```
login12.java X
1 package com.day20;
2
3 public class login12 extends page12 {
4
5     //name and id taken from parent itself.
6
7     public login12() {
8         this.name=name;
9         this.id=id;
10        System.out.println(this.name);
11        System.out.println(this.id);
12        System.out.println("default child constructor");
13    }
14
15    public login12(String name, int id) {
16        this.name=name;
17        this.id=id;
18        System.out.println(this.name);
19        System.out.println(this.id);
20        System.out.println("parameter child constructor");
21    }
22
23    @Override
24    public boolean unloading() {
25        System.out.println("child class override unloading");
26        return false;
27    }
28
29    @Override
30    public void loading() {
31        System.out.println("child class override loading");
32    }
33
34
35}
36
```

The screenshot shows a Java code editor with two tabs at the top: "login12.java" and "testfor12.java". The "testfor12.java" tab is active. The code is as follows:

```
1 package com.day20;
2
3 public class testfor12 {
4     //first parent class default constructor called, then child class
5     //constructor called.
6     //This sequence predefined in java for any inherited class.
7
8     //here parent class doesn't have any constructor so no issues.
9     //everything called from child.
10
11
12 public static void main(String[] args) {
13
14     login12 l1=new login12();
15
16     l1.loading();
17     l1.title();
18     boolean b1=l1.unloading();
19     System.out.println(b1);
20     String s1=l1.url();
21     System.out.println(s1);
22
23     System.out.println("-----");
24
25     login12 l2=new login12("tiger", 543543);
```

```
24
25     login12 l2=new login12("tiger", 543543);
26
27     l1.loading();
28     l1.title();
29     boolean b2=l1.unloading();
30     System.out.println(b1);
31     String s11=l1.url();
32     System.out.println(s11);
33
34 }
35
36 }
```

```
35
36 }
37
38
39 //null
40 //0
41 //default child constructor
42 //child class override loading
43 //parent class login title method
44 //child class override unloading
45 //false
46 //parent class url method
47 //gorilla
48 -----
49 //tiger
50 //543543
51 //parameter child constructor
52 //child class override loading
53 //parent class login title method
54 //child class override unloading
55 //false
56 //parent class url method
57 //gorilla
58
59
60
```

```
--  
59  
60  
61  
62  
63  
64
```

paste page13, login13, testfor13-

```

1 package com.day20;
2
3 public abstract class page13 {
4
5     String name;
6     int id;
7
8
9     //create one abstract method with return.
10    public abstract boolean unloading();
11
12    //create one abstract method without return.
13    public abstract void loading();
14
15    //create normal method with return.
16    public String url() {
17        System.out.println("parent class url method");
18        return "gorilla";
19    }
20
21    //create normal method without return.
22    public void title() {
23        System.out.println("parent class login title method");
24    }
25
26}
27

```

```

1 package com.day20;
2
3 public class login13 extends page12 {
4
5     //name and id taken from parent itself.
6
7     public login13() {
8         //add super here.
9         super(); // no constructor to call hence super has no effect.
10        this.name=name;
11        this.id=id;
12        System.out.println(this.name);
13        System.out.println(this.id);
14        System.out.println("default child constructor");
15    }
16
17    public login13(String name, int id) {
18        super(name, id); //The constructor page12(String, int) is undefined
19        //since there is no parent constructor.
20        this.name=name;
21        this.id=id;
22        System.out.println(this.name);
23        System.out.println(this.id);
24        System.out.println("parameter child constructor");
25    }
26

```

```
24         System.out.println("parameter child constructor");
25     }
26
27     @Override
28     public boolean unloading() {
29         System.out.println("child class override unloading");
30         return false;
31     }
32
33     @Override
34     public void loading() {
35         System.out.println("child class override loading");
36     }
37
38
39 }
40
```

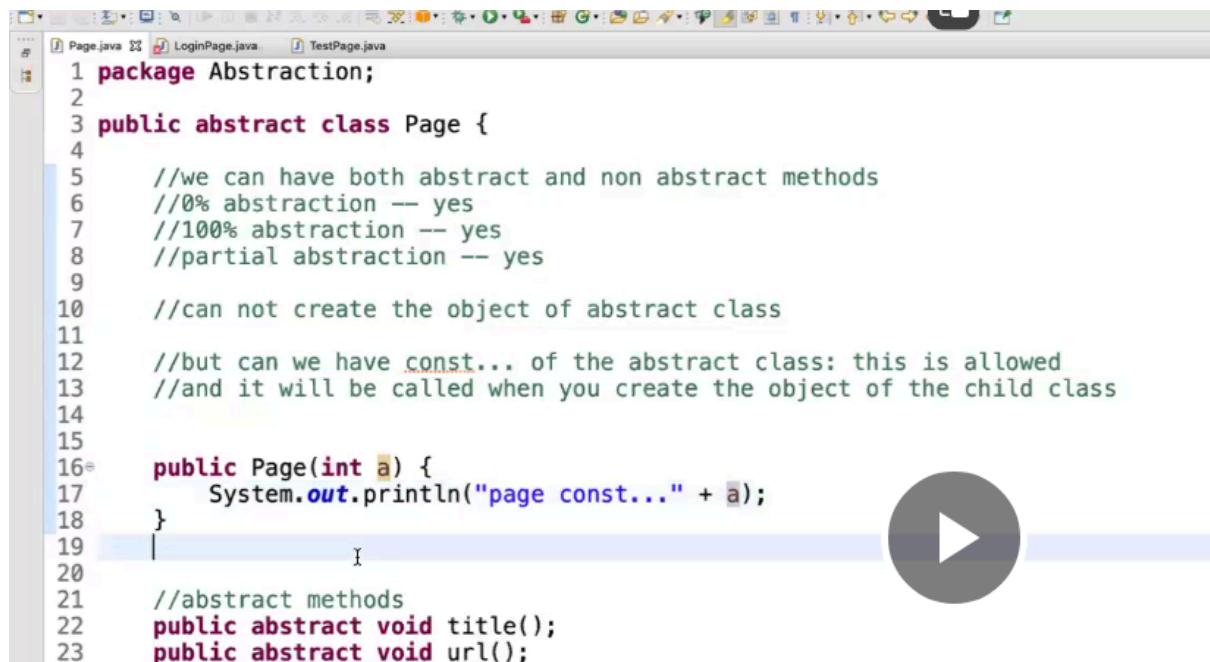


```
testfor13.java X
1 package com.day20;
2
3 public class testfor13 {
4     //first parent class default constructor called, then child class
5     //constructor called.
6     //This sequence predefined in java for any inherited class.
7
8     //here parent class doesn't have any constructor so no issues.
9     //everything called from child.
10
11
12     public static void main(String[] args) {
13
14         login13 l1=new login13();
15
16         l1.loading();
17         l1.title();
18         boolean b1=l1.unloading();
19         System.out.println(b1);
20         String s1=l1.url();
21         System.out.println(s1);
22
23         System.out.println("-----");
24
25         login13 l2=new login13("tiger", 543543);
```

```
24
25     login13 l2=new login13("tiger", 543543);
26
27     l1.loading();
28     l1.title();
29     boolean b2=l1.unloading();
30     System.out.println(b1);
31     String s11=l1.url();
32     System.out.println(s11);
33
34 }
35
36 }
37
38
39 //null
40 //0
41 //default child constructor
42 //child class override loading
43 //parent class login title method
44 //child class override unloading
45 //false
46 //parent class url method
47 //gorilla
48 //
```

```
46 //parent class url method
47 //gorilla
48 //-----
49 //tiger
50 //543543
51 //parameter child constructor
52 //child class override loading
53 //parent class login title method
54 //child class override unloading
55 //false
56 //parent class url method
57 //gorilla
58
59
60
61
62
63
64
65
```

Parent has parameterised constructor-



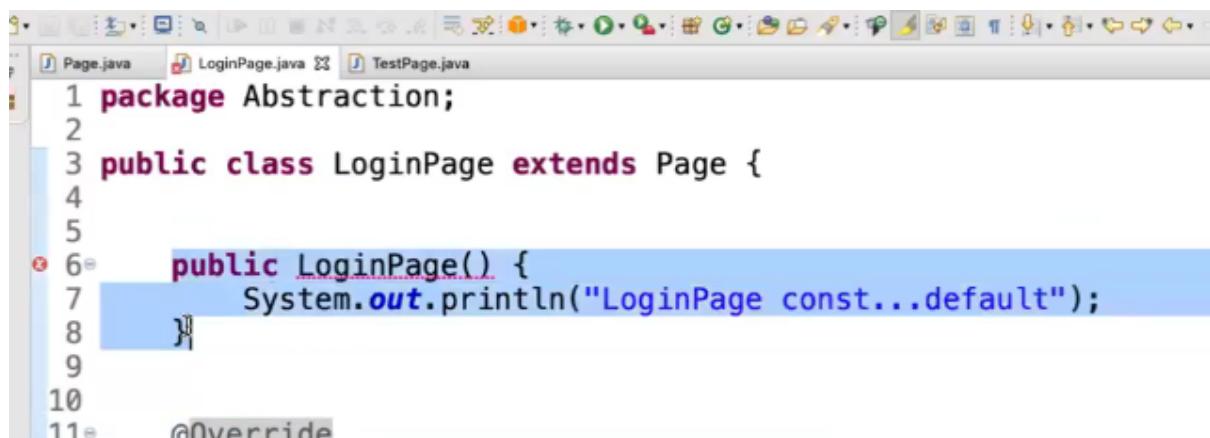
```

1 package Abstraction;
2
3 public abstract class Page {
4
5     //we can have both abstract and non abstract methods
6     //0% abstraction -- yes
7     //100% abstraction -- yes
8     //partial abstraction -- yes
9
10    //can not create the object of abstract class
11
12    //but can we have const... of the abstract class: this is allowed
13    //and it will be called when you create the object of the child class
14
15
16    public Page(int a) {
17        System.out.println("page const..." + a);
18    }
19
20    ...
21
22    //abstract methods
23    public abstract void title();
24    public abstract void url();

```

Login page gives error-`/Implicit super constructor page10() is undefined.`

`//Must explicitly invoke another constructor`



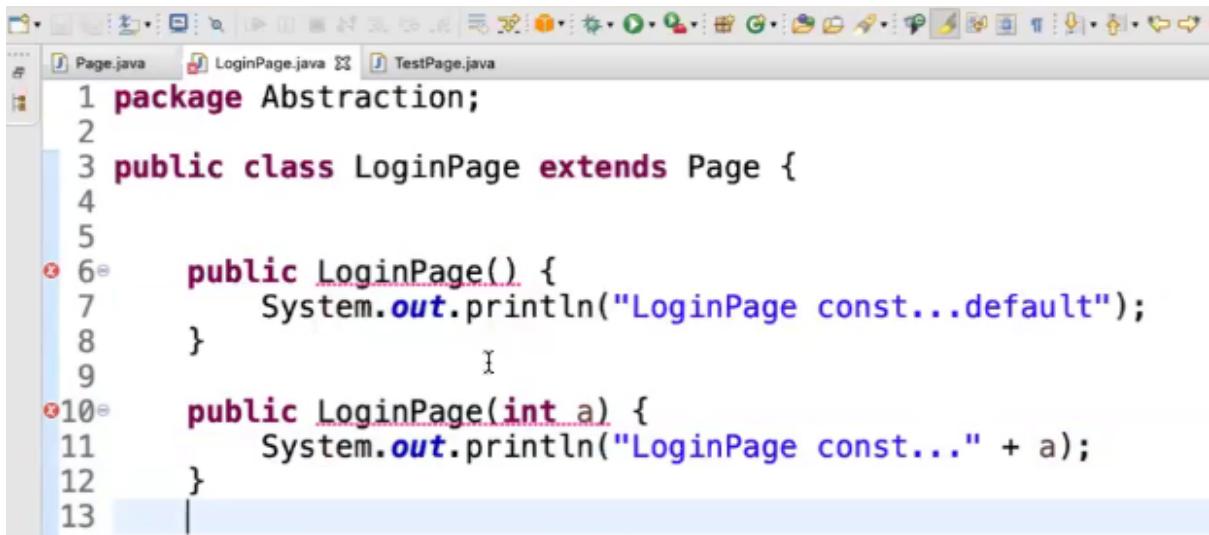
```

1 package Abstraction;
2
3 public class LoginPage extends Page {
4
5
6    public LoginPage() {
7        System.out.println("LoginPage const...default");
8    }
9
10
11    @Override

```

**Even with parameter in child class constructor
same error-`/Implicit super constructor page10() is undefined.`**

`//Must explicitly invoke another constructor`



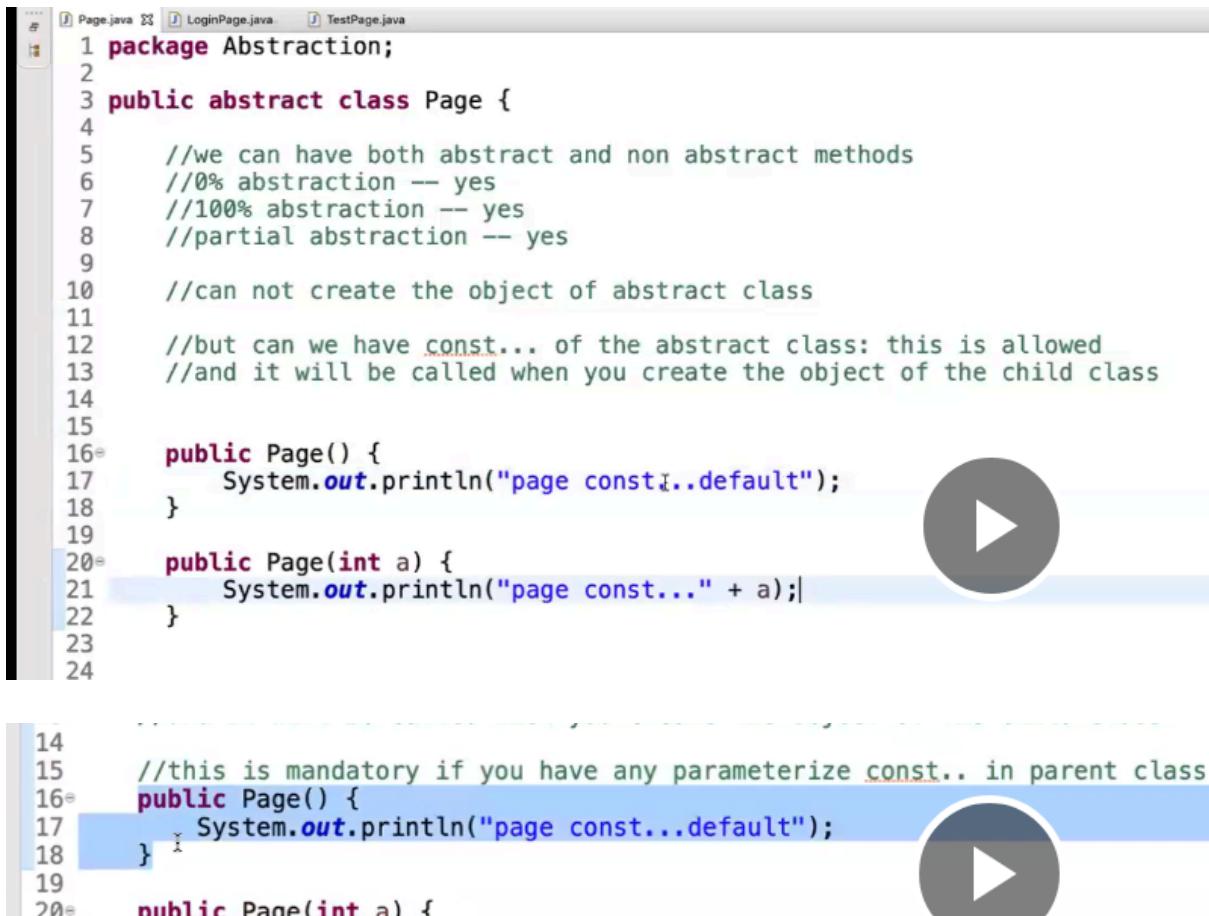
```

1 package Abstraction;
2
3 public class LoginPage extends Page {
4
5
6     public LoginPage() {
7         System.out.println("LoginPage const...default");
8     }
9
10    public LoginPage(int a) {
11        System.out.println("LoginPage const..." + a);
12    }
13

```

It is mandatory now to write default constructor in parent if we wrote parameterised constructor.

Add default and parameterised in parent-

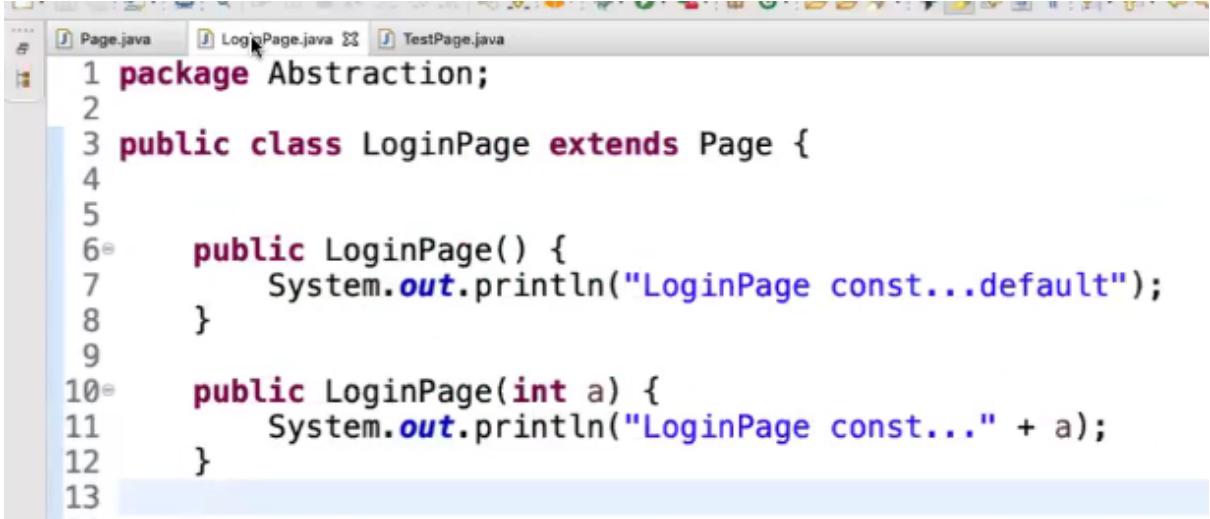


```

1 package Abstraction;
2
3 public abstract class Page {
4
5     //we can have both abstract and non abstract methods
6     //0% abstraction -- yes
7     //100% abstraction -- yes
8     //partial abstraction -- yes
9
10    //can not create the object of abstract class
11
12    //but can we have const... of the abstract class: this is allowed
13    //and it will be called when you create the object of the child class
14
15
16    public Page() {
17        System.out.println("page const...default");
18    }
19
20    public Page(int a) {
21        System.out.println("page const..." + a);
22    }
23
24
14
15    //this is mandatory if you have any parameterize const.. in parent class
16    public Page() {
17        System.out.println("page const...default");
18    }
19
20    public Page(int a) {

```

Login page-

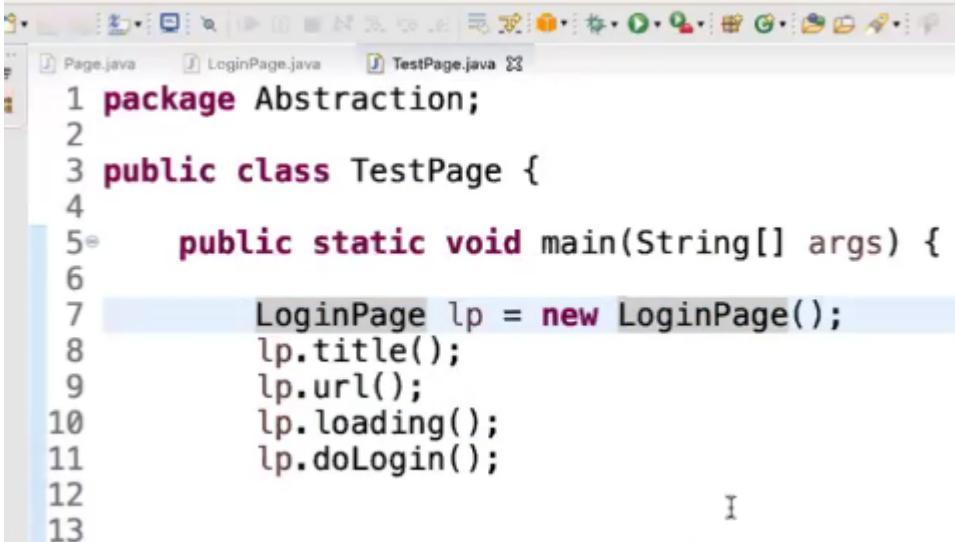


```

1 package Abstraction;
2
3 public class LoginPage extends Page {
4
5
6     public LoginPage() {
7         System.out.println("LoginPage const...default");
8     }
9
10    public LoginPage(int a) {
11        System.out.println("LoginPage const..." + a);
12    }
13}

```

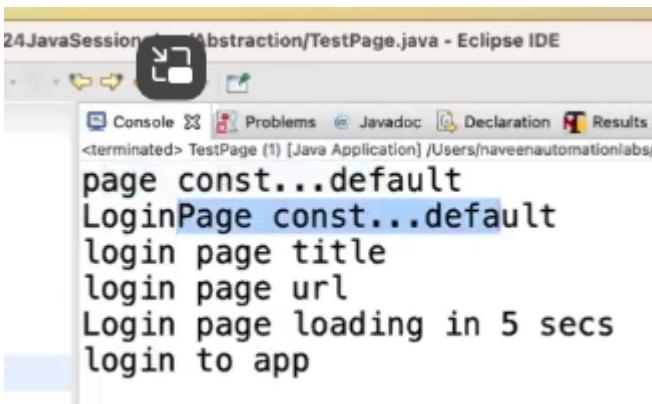
Test-



```

1 package Abstraction;
2
3 public class TestPage {
4
5     public static void main(String[] args) {
6
7         LoginPage lp = new LoginPage();
8         lp.title();
9         lp.url();
10        lp.loading();
11        lp.doLogin();
12
13

```



24JavaSession / Abstraction/TestPage.java - Eclipse IDE

Console Problems Javadoc Declaration Results

<terminated> TestPage (1) [Java Application] /Users/naveenautomationlabs/

```

page const...default
LoginPage const...default
login page title
login page url
Login page loading in 5 secs
login to app

```

Pass param-

The screenshot shows the Eclipse IDE interface. In the top editor area, the file `TestPage.java` is open, displaying the following code:

```

1 package Abstraction;
2
3 public class TestPage {
4
5     public static void main(String[] args) {
6
7         LoginPage lp = new LoginPage(10);
8         lp.title();
9         lp.url();
10        lp.loading();
11        lp.doLogin();
12
13

```

In the bottom console window, the output of the program execution is shown:

```

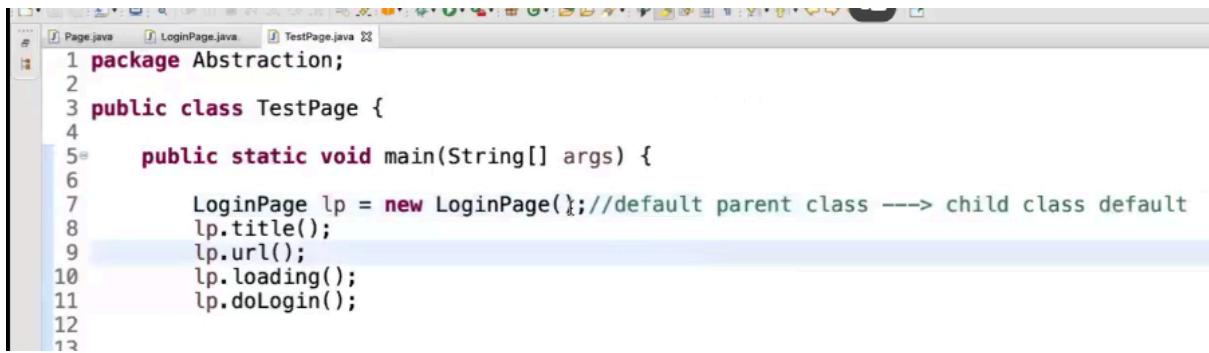
<terminated> TestPage [1] [Java Application] /Users/naveenautomationlabs/p2/poo
page const...default
LoginPage const...10
login page title
login page url
Login page loading in 5 secs
login to app

```

Default from parent called, parameterised one from child called.

Always remember parent class default constructor will be called everytime.

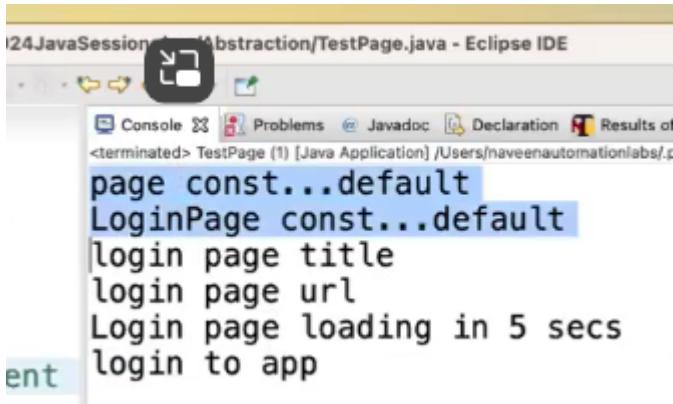
Default is called-
sequence



```

1 package Abstraction;
2
3 public class TestPage {
4
5     public static void main(String[] args) {
6
7         LoginPage lp = new LoginPage(); // default parent class ----> child class default
8         lp.title();
9         lp.url();
10        lp.loading();
11        lp.doLogin();
12
13

```

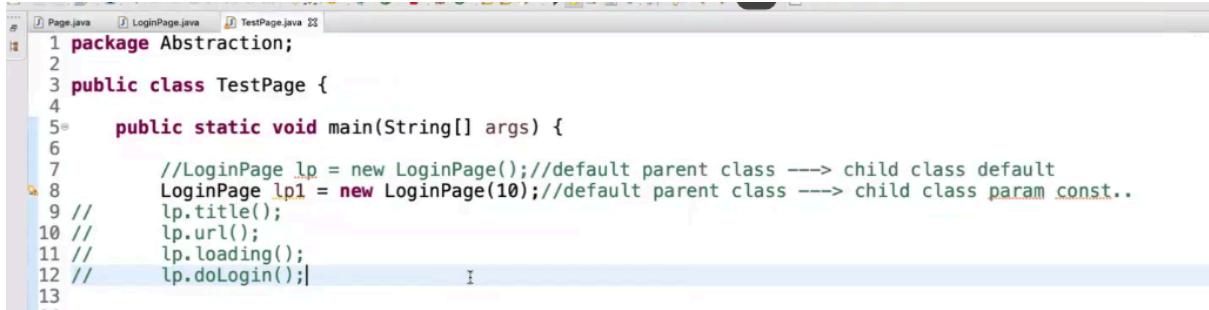


```

page const...default
LoginPage const...default
login page title
login page url
Login page loading in 5 secs
login to app

```

Param constructor- sequence-



```

1 package Abstraction;
2
3 public class TestPage {
4
5     public static void main(String[] args) {
6
7         // LoginPage lp = new LoginPage(); // default parent class ----> child class default
8         LoginPage lp1 = new LoginPage(10); // default parent class ----> child class param const..
9         //
10        lp.title();
11        lp.url();
12        lp.loading();
13        lp.doLogin();

```



```

page const...default
LoginPage const...10

```

Page-

```

23
24  public Page(int a, int b) {
25      System.out.println("page const..." + a+b);
26  }
27

```

Login page-

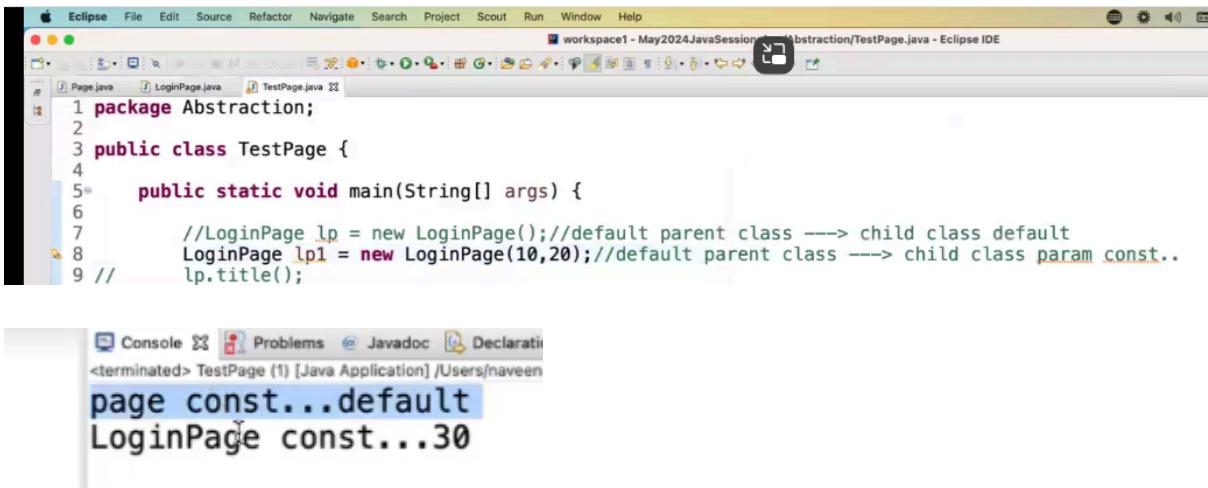
Same in child.

```

12      }
13
14  public LoginPage(int a, int b) {
15      System.out.println("LoginPage const..." + a+b);
16  }
17
18  |
19  @Override

```

Test-



The screenshot shows the Eclipse IDE interface with the following details:

- Top Bar:** Eclipse, File, Edit, Source, Refactor, Navigate, Search, Project, Scout, Run, Window, Help.
- Title Bar:** workspace1 - May2024JavaSession - Abstraction/TestPage.java - Eclipse IDE
- Left Sidebar:** Shows Page.java, LoginPage.java, and TestPage.java.
- Code Editor:**

```

1 package Abstraction;
2
3 public class TestPage {
4
5     public static void main(String[] args) {
6
7         //LoginPage lp = new LoginPage(); //default parent class ---> child class default
8         LoginPage lp1 = new LoginPage(10,20); //default parent class ---> child class param const..
9 //        lp.title();

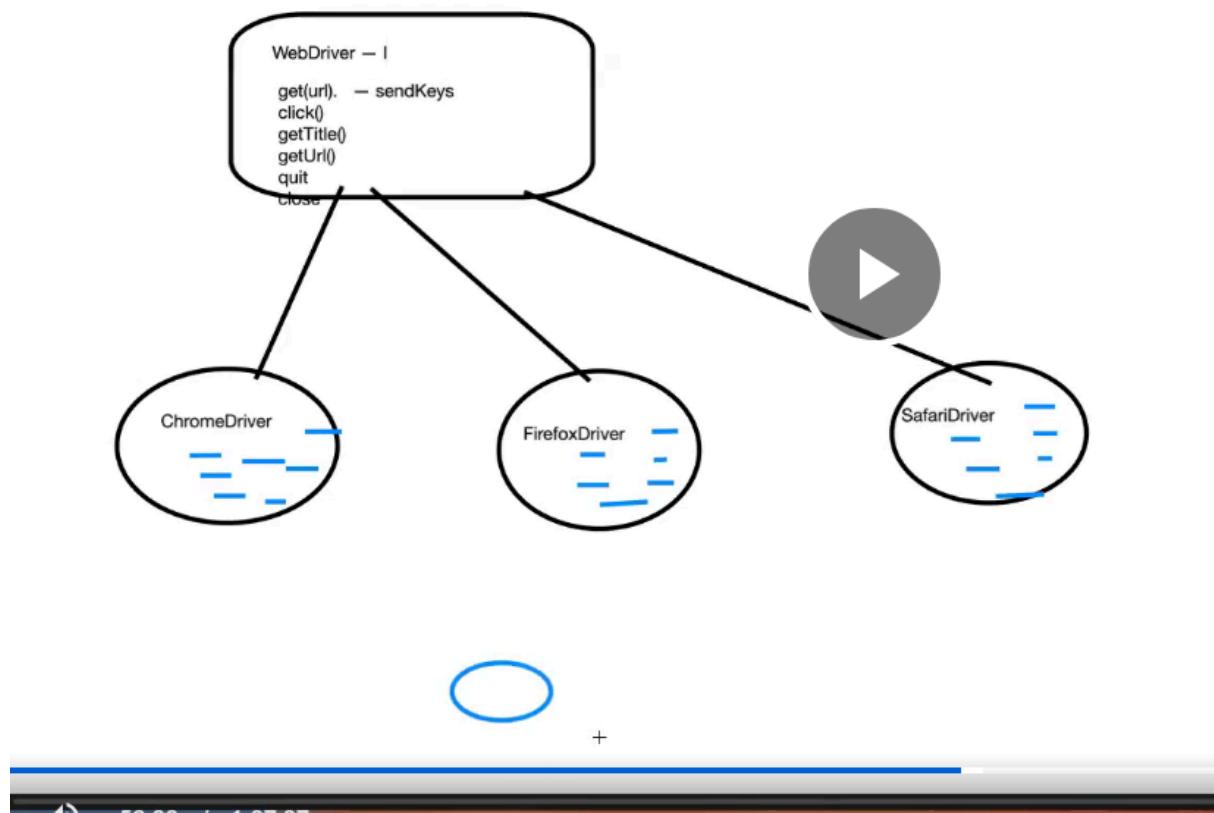
```
- Console Tab:** Shows the output of the program execution.
- Console Output:**

```

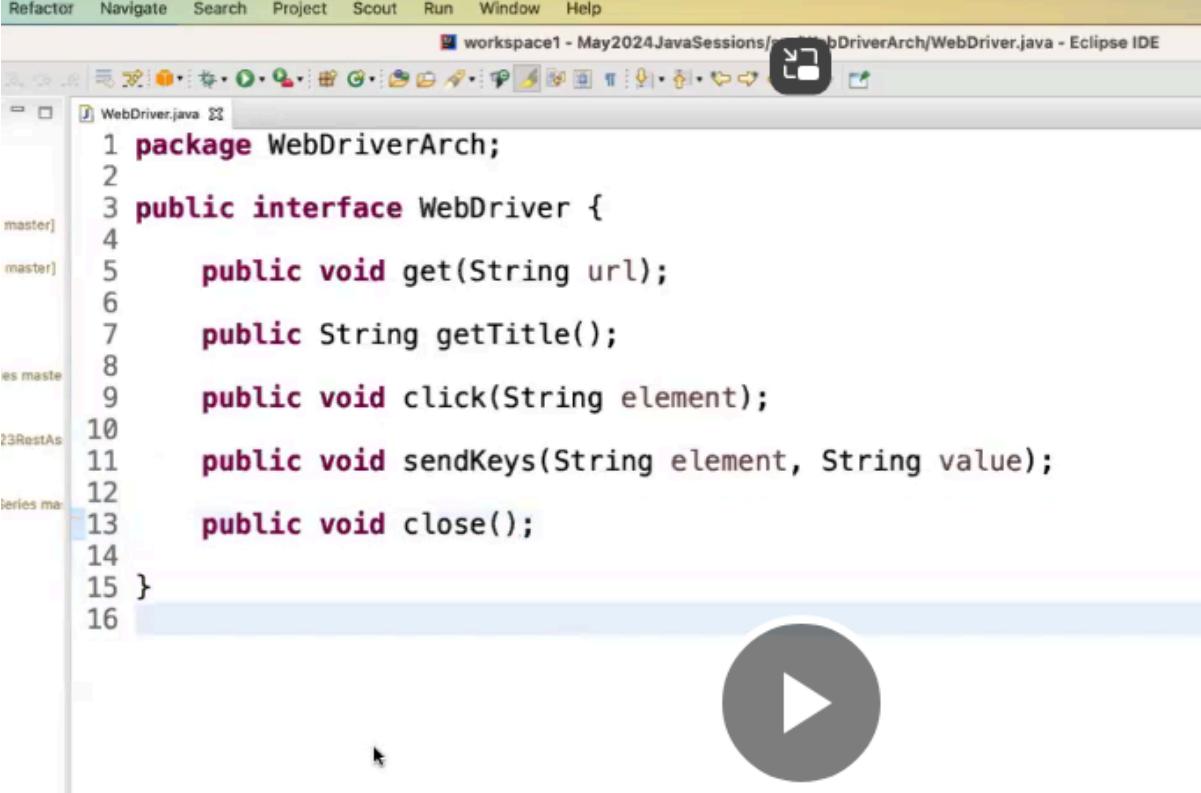
page const...default
LoginPage const...30

```

We will create this-



Webdriver-



The screenshot shows the Eclipse IDE interface with the following details:

- Menu Bar:** Refactor, Navigate, Search, Project, Scout, Run, Window, Help.
- Title Bar:** workspace1 - May2024JavaSessions/WebDriverArch/WebDriver.java - Eclipse IDE
- Toolbar:** Includes icons for file operations, search, project navigation, and code editing.
- Left Sidebar:** Shows a tree view with nodes like "master", "es maste", "23RestAs", and "series ma".
- Code Editor:** Displays the content of `WebDriver.java`. The code defines a public interface `WebDriver` with methods: `get(String url)`, `getTitle()`, `click(String element)`, `sendKeys(String element, String value)`, and `close()`.

```
1 package WebDriverArch;
2
3 public interface WebDriver {
4
5     public void get(String url);
6
7     public String getTitle();
8
9     public void click(String element);
10
11    public void sendKeys(String element, String value);
12
13    public void close();
14
15 }
16
```

Chrome driver-

```
1 package WebDriverArch;
2
3 public class ChromeDriver implements WebDriver {
4
5     public ChromeDriver() {
6         System.out.println("launch chrome browser...");}
7
8
9     @Override
10    public void get(String url) {
11        System.out.println("entering url : " + url);
12    }
13
14    @Override
15    public String getTitle() {
16        return "Google";
17    }
18
19    @Override
20    public void click(String element) {
21        System.out.println("clicking on element . " + element);
22    }
23
24    @Override
25    public void sendKeys(String element, String value) {
26        System.out.println("entering value: " + value + " into field: " + element);
27    }
28
29    @Override
30    public void close() {
31        System.out.println("closing browser");
32    }
33
34 }
```

ff-



```
1 package WebDriverArch;
2
3 public class FirefoxDriver implements WebDriver {
4
5     public FirefoxDriver() {
6         System.out.println("launch firefox browser...");
7     }
8
9     @Override
10    public void get(String url) {
11        System.out.println("entering url : " + url);
12    }
13
14    @Override
15    public String getTitle() {
16        return "Google";
17    }
18
19    @Override
20    public void click(String element) {
21        System.out.println("clicking on element . " + element);
22    }
23
24    @Override
25    public void sendKeys(String element, String value) {
26        System.out.println("entering value: " + value + " into field: " + element);
27    }
28
29    @Override
30    public void close() {
31        System.out.println("closing browser");
32    }
33
34 }
35
```

Safari-



```
1 package WebDriverArch;
2
3 public class SafariDriver implements WebDriver {
4
5     public SafariDriver() {
6         System.out.println("launch safari browser...");
7     }
8
9     @Override
10    public void get(String url) {
11        System.out.println("entering url : " + url);
12    }
13
14    @Override
15    public String getTitle() {
16        return "Google";
17    }
18
19    @Override
20    public void click(String element) {
21        System.out.println("clicking on element . " + element);
22    }
23
24    @Override
25    public void sendKeys(String element, String value) {
26        System.out.println("entering value: " + value + " into field: " + element);
27    }
28
29    @Override
30    public void close() {
31        System.out.println("closing browser");
32    }
33
34 }
```

Test-



```

1 package WebDriverArch;
2
3 public class AmazonTest {
4
5     public static void main(String[] args) {
6
7         ChromeDriver driver = new ChromeDriver();
8
9         driver.get("https://google.com");
10
11        String title = driver.getTitle();
12        System.out.println(title);
13
14        driver.sendKeys("search field", "Naveen automation labs");
15        driver.click("searchIcon");
16
17        driver.close();
18
19
20
21    }
22
23 }
24

```

```

Console Problems Javadoc Declaration Results of running suite Debug
<terminated> AmazonTest (17) [Java Application] /Users/naveenautomationlabs/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_14.0.2
launch chrome browser...
entering url : https://google.com
Google
entering value: Naveen automation labs into field: search field
clicking on element : searchIcon
closing browser

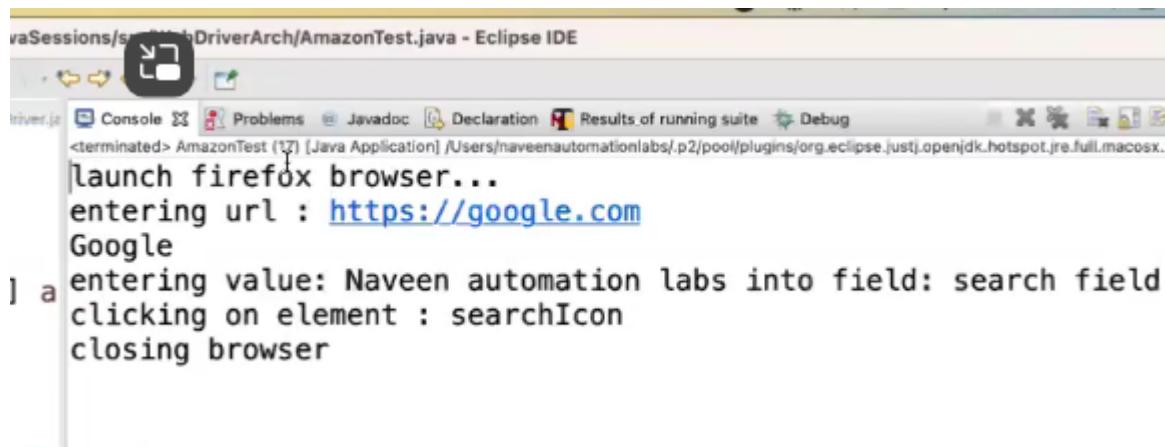
```

Ff-

```

11
12     FirefoxDriver driver = new FirefoxDriver();
13
14
15     driver.get("https://google.com");
16
17     String title = driver.getTitle();
18     System.out.println(title);
19
20     driver.sendKeys("search field", "Naveen automation labs");
21     driver.click("searchIcon");
22
23     driver.close();
24
25
26

```

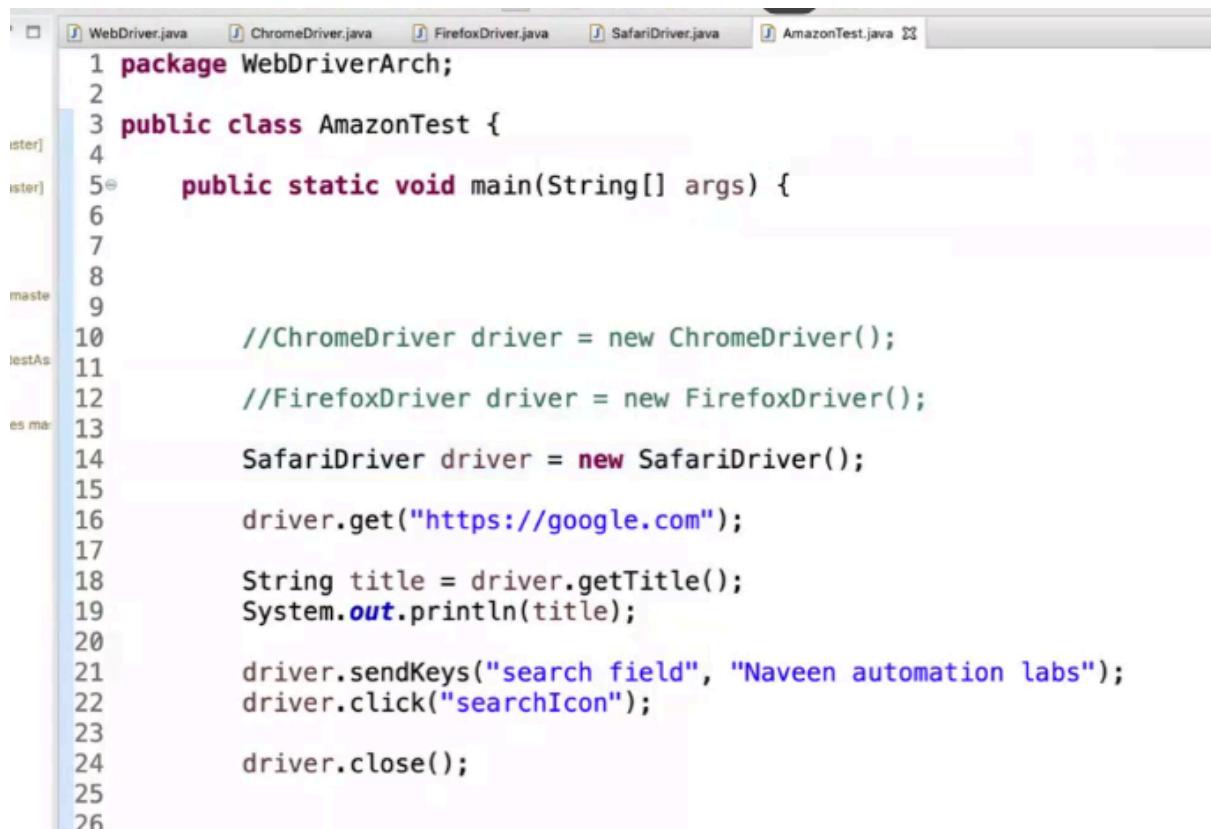


```

vaSessions/sessions/DriverArch/AmazonTest.java - Eclipse IDE
Driver.java
Console Problems Javadoc Declaration Results of running suite Debug
<terminated> AmazonTest (12) [Java Application] /Users/naveenautomationlabs/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx-x86_64
[launch firefox browser...
entering url : https://google.com
Google
] a entering value: Naveen automation labs into field: search field
clicking on element : searchIcon
closing browser

```

Safari-



```

WebDriver.java ChromeDriver.java FirefoxDriver.java SafariDriver.java AmazonTest.java
1 package WebDriverArch;
2
3 public class AmazonTest {
4
5     public static void main(String[] args) {
6
7
8
9
10        //ChromeDriver driver = new ChromeDriver();
11
12        //FirefoxDriver driver = new FirefoxDriver();
13
14        SafariDriver driver = new SafariDriver();
15
16        driver.get("https://google.com");
17
18        String title = driver.getTitle();
19        System.out.println(title);
20
21        driver.sendKeys("search field", "Naveen automation labs");
22        driver.click("searchIcon");
23
24        driver.close();
25
26

```

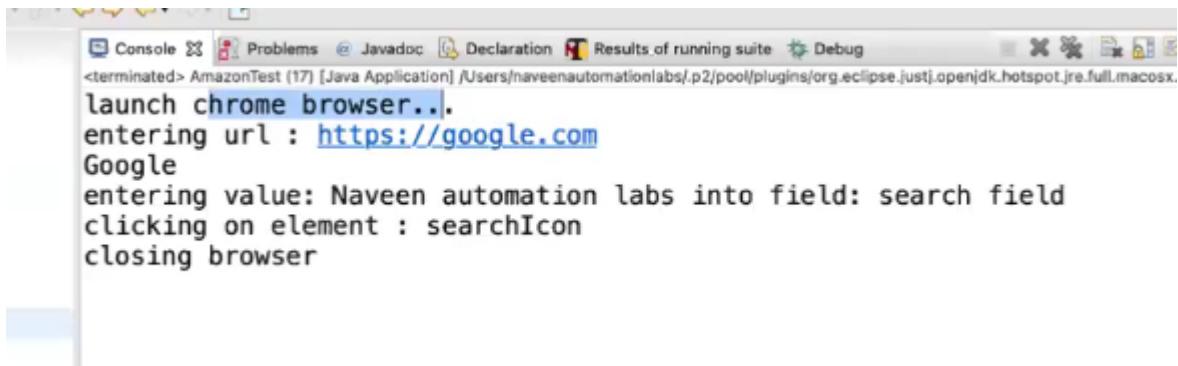
```
Sessions/seleniumDriverArch/AmazonTest.java - Eclipse IDE
[Run] [New Run] [Close]
File Project Run Problems Javadoc Declaration Results of running suite Debug
test.java <terminated> AmazonTest (17) [Java Application] /Users/naveenautomationlabs/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86
launch safari browser...
entering url : https://google.com
Google
a entering value: Naveen automation labs into field: search field
clicking on element : searchIcon
closing browser
```

Problem-

Every time we want to run on browser, we have to comment and uncomment. Top cast can solve the problem.



```
5 public static void main(String[] args) {  
6     //ChromeDriver driver = new ChromeDriver();  
7     //FirefoxDriver driver = new FirefoxDriver();  
8     //SafariDriver driver = new SafariDriver();  
9  
10    String browserName = "chrome";  
11    WebDriver driver = null;  
12  
13    switch (browserName.trim().toLowerCase()) {  
14        case "chrome":  
15            driver = new ChromeDriver();  
16            break;  
17        case "firefox":  
18            driver = new FirefoxDriver();  
19            break;  
20        case "safari":  
21            driver = new SafariDriver();  
22            break;  
23  
24        default:  
25            System.out.println("plz pass the right browser name : " + browserName);  
26            break;  
27    }  
28  
29    driver.get("https://google.com");  
30  
31  
32
```



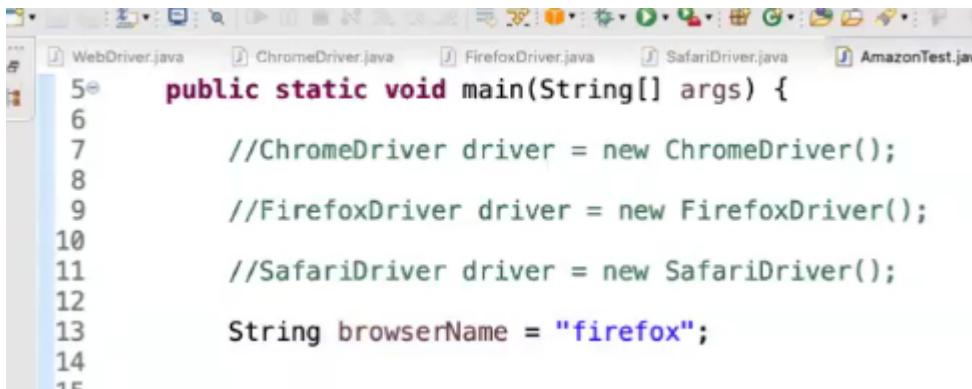
```

Console Problems Javadoc Declaration Results of running suite Debug
<terminated> AmazonTest (17) [Java Application] /Users/naveenautomationlabs/p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx
launch chrome browser...
entering url : https://google.com
Google
entering value: Naveen automation labs into field: search field
clicking on element : searchIcon
closing browser

```

change browser name-

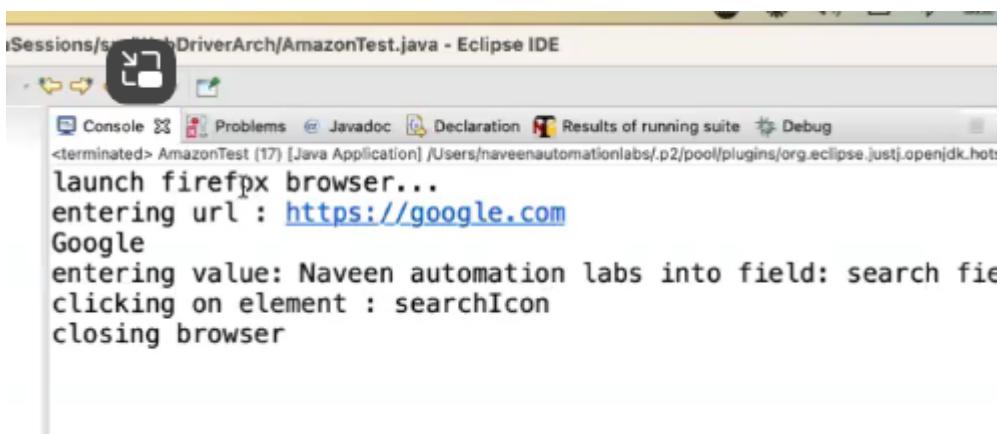
line 13-



```

public static void main(String[] args) {
    //ChromeDriver driver = new ChromeDriver();
    //FirefoxDriver driver = new FirefoxDriver();
    //SafariDriver driver = new SafariDriver();
    String browserName = "firefox";
}

```

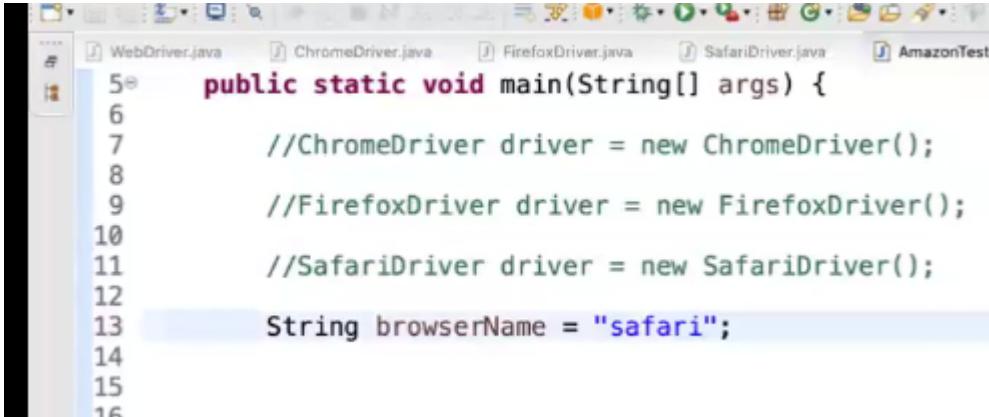


```

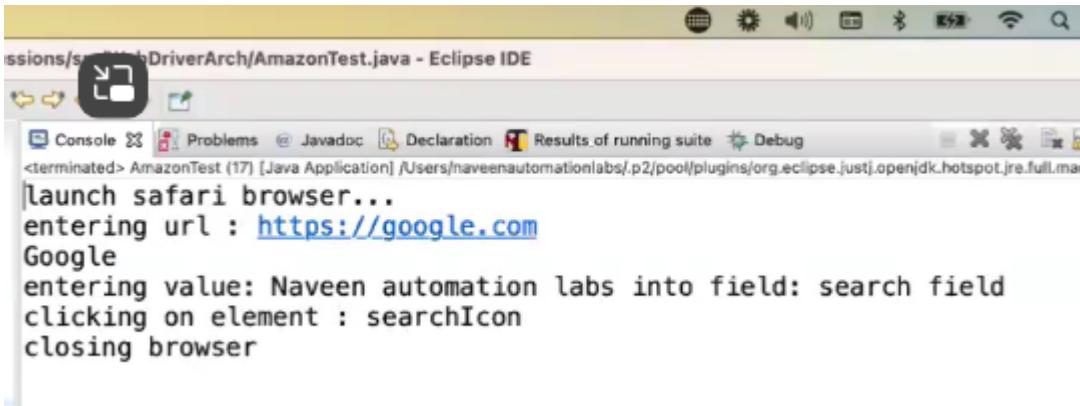
Console Problems Javadoc Declaration Results of running suite Debug
<terminated> AmazonTest (17) [Java Application] /Users/naveenautomationlabs/p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx
launch firepx browser...
entering url : https://google.com
Google
entering value: Naveen automation labs into field: search file
clicking on element : searchIcon
closing browser

```

change line 13 browser name-



```
5*     public static void main(String[] args) {  
6  
7         //ChromeDriver driver = new ChromeDriver();  
8  
9         //FirefoxDriver driver = new FirefoxDriver();  
10  
11        //SafariDriver driver = new SafariDriver();  
12  
13        String browserName = "safari";  
14  
15  
16
```

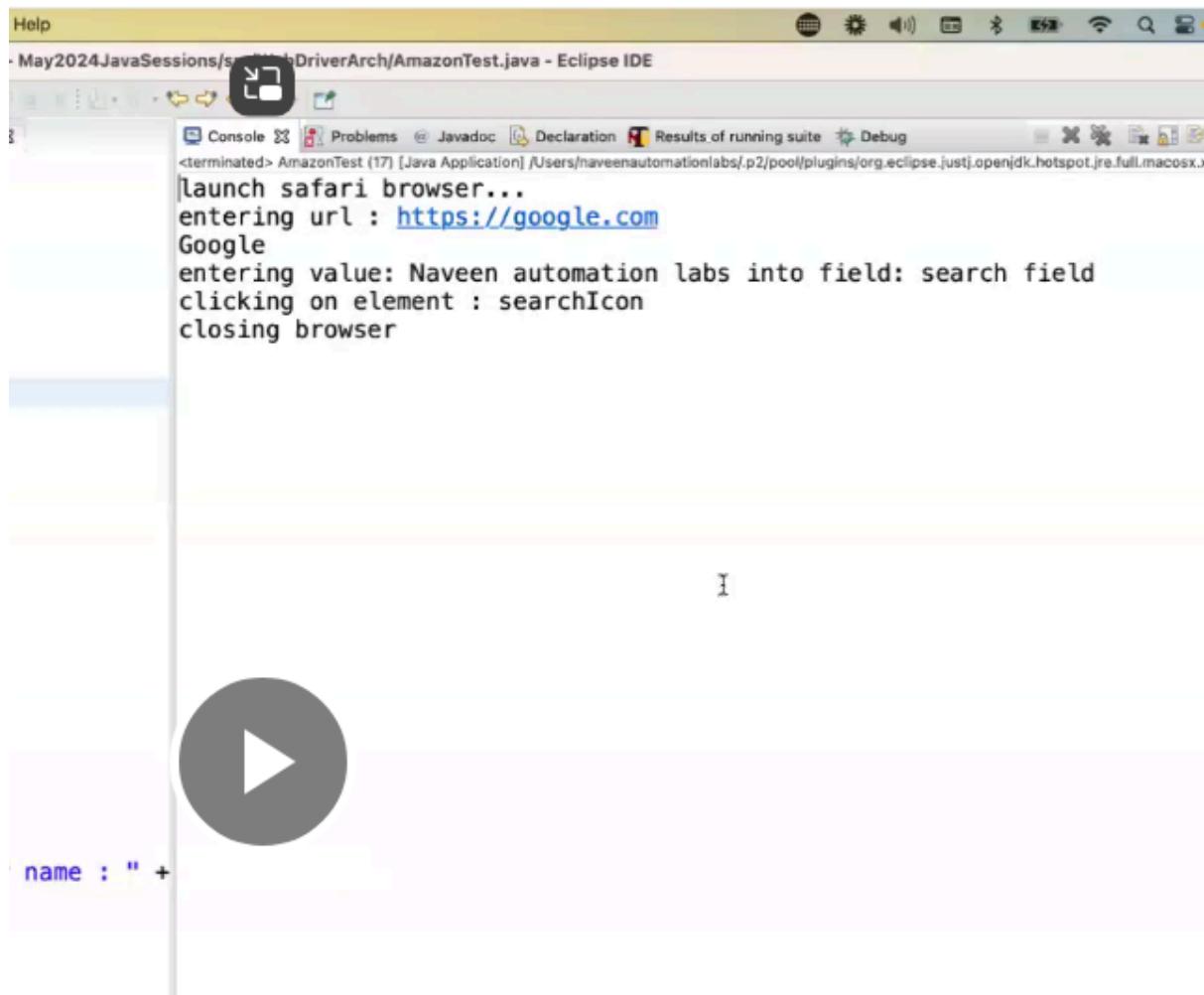


essions/seleniumDriverArch/AmazonTest.java - Eclipse IDE

Console Problems Javadoc Declaration Results of running suite Debug

<terminated> AmazonTest (17) [Java Application] /Users/naveenautomationlabs/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.main

launch safari browser...
entering url : <https://google.com>
Google
entering value: Naveen automation labs into field: search field
clicking on element : searchIcon
closing browser



The screenshot shows the Eclipse IDE interface with a terminal window open. The terminal window title is "May2024JavaSessions/selenium WebDriverArch/AmazonTest.java - Eclipse IDE". The output in the terminal is:

```
<terminated> AmazonTest (17) [Java Application] /Users/naveenautomationlabs/p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64/bin/java -jar selenium-server-standalone-4.10.0.jar &
[INFO] Launching browser...
[INFO] entering url : https://google.com
[INFO] Google
[INFO] entering value: Naveen automation labs into field: search field
[INFO] clicking on element : searchIcon
[INFO] closing browser
```

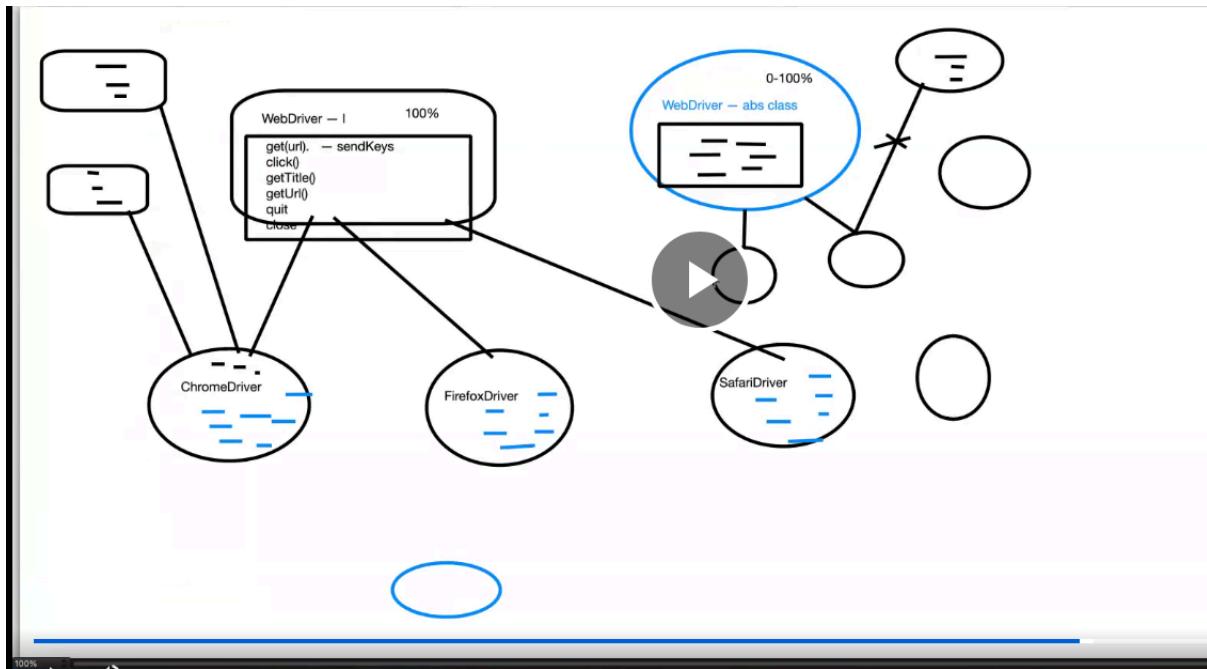
Below the terminal window, there is a large play button icon.

example of real time top casting- cross browser logic-

```
14
15
16     //cross browser logic: top casting
17     WebDriver driver = null;
18     switch (browserName.trim().toLowerCase()) {
19         case "chrome":
20             driver = new ChromeDriver();
21             break;
22         case "firefox":
23             driver = new FirefoxDriver();
24             break;
25         case "safari":
26             driver = new SafariDriver();
27             break;
28
29         default:
30             System.out.println("plz pass the right browser name : " + browserName);
31             break;
32     }
33
```



When to use interface and when to use abstract-



interface -

Multiple inheritance possible.

Methods have to be implemented by child class ensuring consistency.

100 percent abstraction in interface but abstract (may or may not).

Note-

When local variable is declared, we need to initialise it before using, else when we use, we get error.

Example, webdriver driver=null;

Default method in interface with body-

```
15  
16  default void getLogs() {  
17      System.out.println("get logs");  
18  }  
19  
20  
21  m
```

Test class-

```
44  
45      driver.getLogs();  
46  
47
```

Get logs.

paste webdriver3, chromedriver3, ffdriver3,
safaridriver3, testforwebdriver3-

```
1 package com.day20;
2
3 public interface webdriver3 {
4
5     public void get(String url);
6
7     public String getTitle();
8
9     public void click(String element);
10
11    public void sendKeys(String element, String value);
12
13    public void close();
14
15    //default method in webdriver.
16    default void getlogs() {
17        System.out.println("get logs");
18    }
19
20    //create default method with parameter and return type.
21    default String getlocations(String url) {
22        System.out.println("get locations");
23        return url;
24    }
25
26    //default method which is static.
27    //Illegal combination of modifiers for the interface method getnames;
28    //only one of abstract, default, or static permitted
29    // default static String getnames(String name) {
30    //     System.out.println("get names");
31    //     return name;
32    // }
33
34 }
35
```

```

chromedriver3.java
1 package com.day20;
2
3 public class chromedriver3 implements webdriver3 {
4
5     //constructor.
6     public chromedriver3() {
7         System.out.println("default constructor of chrome driver class");
8     }
9
10    @Override
11    public void get(String url) {
12        System.out.println("Launching Chrome browser and navigating to: " + url);
13    }
14
15    @Override
16    public String getTitle() {
17        System.out.println("Getting the page title from Chrome browser...");
18        return "Sample Chrome Page Title";
19    }
20
21    @Override
22    public void click(String element) {
23        System.out.println("Clicking on element: " + element + " in Chrome browser");
24    }
25
26    @Override
27    public void sendKeys(String element, String value) {
28        System.out.println("Typing '" + value + "' into element: " + element + " in Chrome browser");
29    }
30
31    @Override
32    public void close() {
33        System.out.println("Closing Chrome browser");
34    }
35
36 }
37
38 
```



```

ffdriver3.java
1 package com.day20;
2
3 public class ffdriver3 implements webdriver3 {
4
5     //constructor.
6     public ffdriver3() {
7         System.out.println("default constructor of ff driver class");
8     }
9
10    @Override
11    public void get(String url) {
12        System.out.println("Launching ff browser and navigating to: " + url);
13    }
14
15    @Override
16    public String getTitle() {
17        System.out.println("Getting the page title from ff browser...");
18        return "Sample ff Page Title";
19    }
20
21    @Override
22    public void click(String element) {
23        System.out.println("Clicking on element: " + element + " in ff browser");
24    }
25
26    @Override
27    public void sendKeys(String element, String value) {
28        System.out.println("Typing '" + value + "' into element: " + element + " in ff browser");
29    }
30
31 } 
```

```
28     System.out.println("Typing '" + value + "' into element: " + element + " in ff browser");
29 }
30
31 @Override
32 public void close() {
33     System.out.println("Closing ff browser");
34 }
35
36
37 }
38
```



```
1 package com.day20;
2
3 public class safaridriver3 implements webdriver3 {
4
5     //constructor.
6     public safaridriver3() {
7         System.out.println("default constructor of safari driver class");
8     }
9
10    @Override
11    public void get(String url) {
12        System.out.println("Launching safari browser and navigating to: " + url);
13    }
14
15    @Override
16    public String getTitle() {
17        System.out.println("Getting the page title from safari browser...");
18        return "Sample ff Page Title";
19    }
20
21    @Override
22    public void click(String element) {
23        System.out.println("Clicking on element: " + element + " in safari browser");
24    }
25
26    @Override
27    public void sendKeys(String element, String value) {
28        System.out.println("Typing '" + value + "' into element: " + element + " in safari browser");
29    }
30
31    public void sendKeys(String element, String value) {
32        System.out.println("Typing '" + value + "' into element: " + element + " in safari browser");
33    }
34
35
36
37 }
38
```

```
testforwebdriver3.java
1 package com.day20;
2
3 public class testforwebdriver3 {
4
5     public static void main(String[] args) {
6
7         //top casting solves manual changes.
8         String browsername ="chrome";
9         webdriver3 driver=null;
10
11        switch(browsername.trim().toLowerCase()){
12            case "chrome":
13                driver=new chromedriver3();
14                break;
15            case "ff":
16                driver=new ffdriver3();
17                break;
18            case "safari":
19                driver=new safaridriver3();
20                break;
21            default:
22                System.out.println("invalid browser name " + browsername);
23                break;
24            }
25
26            break;
27        }
28
29        //can call default methods from interface directly.
30        driver.getlogs();
31        String s1=driver.getlocations("tatanka");
32        System.out.println(s1);
33    }
34
35    //default constructor of chrome driver class
36    //get logs
37    //get locations
38    //tatanka
39
40
41
42
43
44
45
```

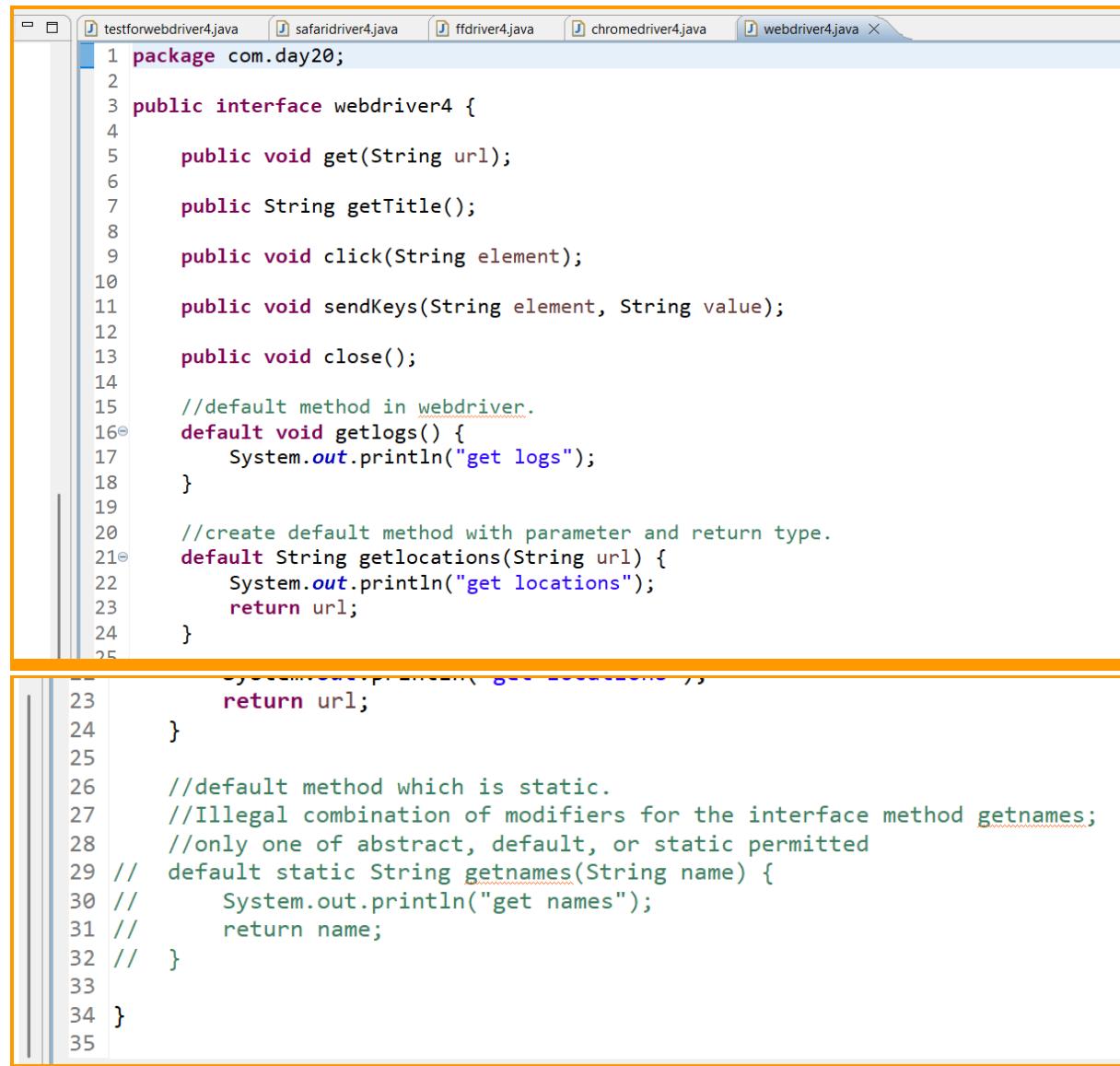
Chrome driver-

Can be inherited also – default methods.

```

29@  @Override
30  public void close() {
31      System.out.println("closing browser");
32      getLogs();
33  }
34
35 }
```

paste webdriver4, chromedriver4, ffdriver4,
safaridriver4, testforwebdriver4-



The screenshot shows an IDE interface with multiple tabs at the top: testforwebdriver4.java, safaridriver4.java, ffdriver4.java, chromedriver4.java, and webdriver4.java X. The code editor displays the following Java code:

```

1 package com.day20;
2
3 public interface webdriver4 {
4
5     public void get(String url);
6
7     public String getTitle();
8
9     public void click(String element);
10
11    public void sendKeys(String element, String value);
12
13    public void close();
14
15    //default method in webdriver.
16    default void getlogs() {
17        System.out.println("get logs");
18    }
19
20    //create default method with parameter and return type.
21    default String getlocations(String url) {
22        System.out.println("get locations");
23        return url;
24    }
25
26    //return url;
27
28    //default method which is static.
29    //Illegal combination of modifiers for the interface method getnames;
30    //only one of abstract, default, or static permitted
31    // default static String getnames(String name) {
32    //     System.out.println("get names");
33    //     return name;
34    //}
35 }
```

```

testforwebdriver4.java  safaridriver4.java  ffdriver4.java  chromedriver4.java
1 package com.day20;
2
3 public class chromedriver4 implements webdriver4 {
4
5     //constructor.
6     public chromedriver4() {
7         System.out.println("default constructor of chrome driver class");
8     }
9
10    @Override
11    public void get(String url) {
12        System.out.println("Launching Chrome browser and navigating to: " + url);
13    }
14
15    @Override
16    public String getTitle() {
17        System.out.println("Getting the page title from Chrome browser...");
18        return "Sample Chrome Page Title";
19    }
20
21    @Override
22    public void click(String element) {
23        System.out.println("Clicking on element: " + element + " in Chrome browser");
24        getlogs();
25    }
26
27        getlogs();
28    }
29
30    @Override
31    public void sendKeys(String element, String value) {
32        System.out.println("Typing '" + value + "' into element: " + element + " in Chrome browser");
33    }
34
35    @Override
36    public void close() {
37        System.out.println("Closing Chrome browser");
38        String s1=getlocations("kanada");
39        System.out.println(s1);
40    }
41

```



```

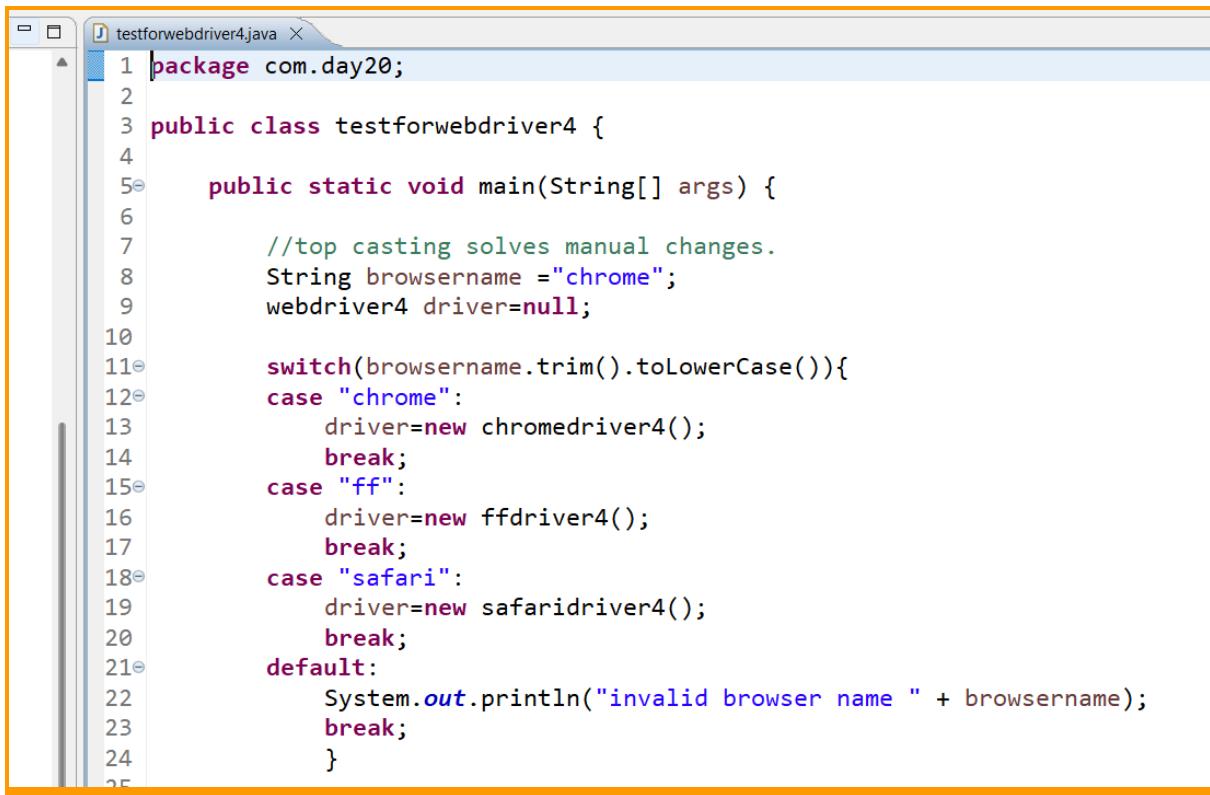
testforwebdriver4.java  safaridriver4.java  ffdriver4.java
1 package com.day20;
2
3 public class ffdriver4 implements webdriver4 {
4
5     //constructor.
6     public ffdriver4() {
7         System.out.println("default constructor of ff driver class");
8     }
9
10    @Override
11    public void get(String url) {
12        System.out.println("Launching ff browser and navigating to: " + url);
13    }
14
15    @Override
16    public String getTitle() {
17        System.out.println("Getting the page title from ff browser...");
18        return "Sample ff Page Title";
19    }
20
21    @Override
22    public void click(String element) {
23        System.out.println("Clicking on element: " + element + " in ff browser");
24    }
25

```

```
23     System.out.println("Clicking on element: " + element + " in ff browser");
24 }
25
26 @Override
27 public void sendKeys(String element, String value) {
28     System.out.println("Typing '" + value + "' into element: " + element + " in ff browser");
29 }
30
31 @Override
32 public void close() {
33     System.out.println("Closing ff browser");
34 }
35
36
37 }
38
```



```
1 package com.day20;
2
3 public class safaridriver4 implements webdriver4 {
4
5     //constructor.
6     public safaridriver4() {
7         System.out.println("default constructor of safari driver class");
8     }
9
10    @Override
11    public void get(String url) {
12        System.out.println("Launching safari browser and navigating to: " + url);
13    }
14
15    @Override
16    public String getTitle() {
17        System.out.println("Getting the page title from safari browser...");
18        return "Sample ff Page Title";
19    }
20
21    @Override
22    public void click(String element) {
23        System.out.println("Clicking on element: " + element + " in safari browser");
24    }
25
26    @Override
27    public void sendKeys(String element, String value) {
28        System.out.println("Typing '" + value + "' into element: " + element + " in safari browser");
29    }
30
31        System.out.println("Typing '" + value + "' into element: " + element + " in safari browser");
32 }
33
34 @Override
35 public void close() {
36     System.out.println("Closing safari browser");
37 }
38
```



The screenshot shows a Java code editor window with an orange border. The file is named "testforwebdriver4.java". The code implements a switch statement to handle different browser names:

```
1 package com.day20;
2
3 public class testforwebdriver4 {
4
5     public static void main(String[] args) {
6
7         //top casting solves manual changes.
8         String browsername ="chrome";
9         webdriver4 driver=null;
10
11     switch(browsername.trim().toLowerCase()){
12         case "chrome":
13             driver=new chromedriver4();
14             break;
15         case "ff":
16             driver=new ffdriver4();
17             break;
18         case "safari":
19             driver=new safaridriver4();
20             break;
21     default:
22         System.out.println("invalid browser name " + browsername);
23         break;
24     }
25 }
```

```
23     break;
24 }
25
26 driver.get("https://www.google.com/");
27 String title=driver.getTitle();
28 System.out.println(title);
29 driver.sendKeys("search chrome", "chrome browser search is happening");
30 driver.click("search chrome");
31 driver.close();
32
33 }
34
35 }
36
37 //default constructor of chrome driver class
38 //Launching Chrome browser and navigating to: https://www.google.com/
39 //Getting the page title from Chrome browser...
40 //Sample Chrome Page Title
41 //Typing 'chrome browser search is happening' into element: search chrome in Chrome browser
42 //Clicking on element: search chrome in Chrome browser
43 //get logs
44 //Closing Chrome browser
```

```
43 //get logs
44 //Closing Chrome browser
45 //get locations
46 //kanada
47
48
49
50
51
52
53
54
55
```