

A decorative graphic on the left side of the slide consisting of two overlapping parallelograms. The front one is blue and the back one is a light green color. They are positioned diagonally, with the blue one partially covering the green one.

Creational Design Patterns

By Raza Sikander
Project Engineer
CDAC Hyderabad



Creational Design Pattern

- Creational design patterns are concerned with the way of creating objects.
- These design patterns are used when a decision must be made at the time of instantiation of a class



Creational Design Pattern

- Singleton Pattern
- Builder Pattern
- Factory Method Pattern
- Abstract Factory Pattern
- Prototype Pattern



Singleton Pattern

“Define a class that has only one instance and provides a global point of access to it”



Singleton Pattern

“Define a class that has only one instance and provides a global point of access to it”

In other words, a class must ensure that only single instance should be created and single object can be used by all other classes.



Singleton Pattern

There are two forms of singleton design pattern

- Early Instantiation: creation of instance at load time
- Lazy Instantiation: creation of instance when required



Singleton Pattern

Advantage of Singleton design pattern

- Saves memory because object is not created at each request. Only single instance is reused again and again.



Singleton Pattern

Usage of Singleton design pattern

- Singleton pattern is mostly used in multi-threaded and database applications. It is used in logging, caching, thread pools, configuration settings etc.



Singleton Pattern

Early Instantiation

```
1. package DesignPatterns;

2. public class Singleton {
3.     private static Singleton instance = new Singleton(); //Eager instantiated
4.
5.     private Singleton()
6.     {
7.         System.out.println("Singleton being initialized");
8.     }
9.
10.    public static Singleton getInstance()
11.    {
12.        return instance;
13.    }
14. }
```



Singleton Pattern

Lazy Instantiation

```
1. package DesignPatterns;

2. public class Singleton {
3.     private static Singleton instance = null;
4.
5.     private Singleton()
6.     {
7.         System.out.println("Singleton being initialized");
8.     }
9.
10.    public static Singleton getInstance() //Only instantiated when getInstance is invoked
11.    {
12.        if(instance == null)
13.            instance = new Singleton();
14.        return instance;
15.    }
16. }
```



Singleton Pattern

Implementation of Singleton Pattern Example

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

Start Page X DesignPattern.java X SingletonPattern.java X

Source History

```
5
6 package designpattern;
7
8 /**
9  *
10  * @author shaik
11  */
12
13 class Singleton
14 {
15     // static variable single_instance of type Singleton
16     private static Singleton single_instance = null;
17
18     // variable of type String
19     public String s;
20
21     // private constructor restricted to this class itself
22     private Singleton()
23     {
24         s = "Hello I am a string part of Singleton class";
25     }
26
27     // static method to create instance of Singleton class
28     public static Singleton getInstance()
29     {
30         if (single_instance == null)
31             single_instance = new Singleton();
32
33         return single_instance;
34     }
35 }
```

designpattern.Singleton

Output Navigator

34:11 INS

Type here to search

ENG IN 12:17 12-11-2019

Start Page x DesignPattern.java x SingletonPattern.java x

Source History

```
36
37 public class SingletonPattern{
38     public static void main(String args[])
39     {
40         // instantiating Singleton class with variable x
41         Singleton x = Singleton.getInstance();
42         Singleton y = Singleton.getInstance();
43         Singleton z = Singleton.getInstance();
44         //Print hash code of the object
45         System.out.println(x.hashCode());
46         System.out.println(y.hashCode());
47         System.out.println(z.hashCode());
48
49
50         // changing variable of instance x
51         x.s = (x.s).toUpperCase();
52
53         System.out.println("String from x is " + x.s);
54         System.out.println("String from y is " + y.s);
55         System.out.println("String from z is " + z.s);
56         System.out.println("\n");
57
58         // changing variable of instance z
59         z.s = (z.s).toLowerCase();
60
61         System.out.println("String from x is " + x.s);
62         System.out.println("String from y is " + y.s);
63         System.out.println("String from z is " + z.s);
64     }
65 }
66
```

designpattern.SingletonPattern >

Output Navigator

37:1

INS



Builder Pattern

"Construct a complex object from simple objects using step-by-step approach"



Builder Pattern

"Construct a complex object from simple objects using step-by-step approach"

It is mostly used when object can't be created in single step like in the de-serialization of a complex object.



Builder Pattern

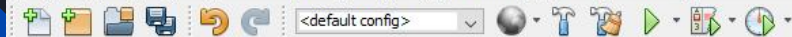
The main advantages of Builder Pattern are as follows:

- It provides clear separation between the construction and representation of an object.
- It provides better control over construction process.
- It supports to change the internal representation of objects.



Builder Pattern

Implementation of Builder Pattern Example



Start Page x DesignPattern.java x SingletonPattern.java x BuilderPattern1.java x BuilderPattern2.java x

Source History

```
13 public class BuilderPattern1 {
14
15     //required parameters
16     private String HDD;
17     private String RAM;
18
19     //optional parameters
20     private boolean isGraphicsCardEnabled;
21     private boolean isBluetoothEnabled;
22
23     public String getHDD() {
24         return HDD;
25     }
26     public String getRAM() {
27         return RAM;
28     }
29     public boolean isGraphicsCardEnabled() {
30         return isGraphicsCardEnabled;
31     }
32     public boolean isBluetoothEnabled() {
33         return isBluetoothEnabled;
34     }
35     private BuilderPattern1(ComputerBuilder builder) {
36         this.HDD=builder.HDD;
37         this.RAM=builder.RAM;
38         this.isGraphicsCardEnabled=builder.isGraphicsCardEnabled;
39         this.isBluetoothEnabled=builder.isBluetoothEnabled;
40     }
41     @Override
42     public String toString() {
```

designpattern.BuilderPattern1 > getHDD >

Output Navigator

23:33

INS



DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

Start Page x DesignPattern.java x SingletonPattern.java x BuilderPattern1.java x BuilderPattern2.java x

Source History

```
46 //Builder Class
47 public static class ComputerBuilder{
48
49     // required parameters
50     private String HDD;
51     private String RAM;
52
53     // optional parameters
54     private boolean isGraphicsCardEnabled;
55     private boolean isBluetoothEnabled;
56
57     public ComputerBuilder(String hdd, String ram){
58         this.HDD=hdd;
59         this.RAM=ram;
60     }
61
62     public ComputerBuilder setGraphicsCardEnabled(boolean isGraphicsCardEnabled) {
63         this.isGraphicsCardEnabled = isGraphicsCardEnabled;
64         return this;
65     }
66
67     public ComputerBuilder setBluetoothEnabled(boolean isBluetoothEnabled) {
68         this.isBluetoothEnabled = isBluetoothEnabled;
69         return this;
70     }
71
72     public BuilderPattern1 build(){
73         return new BuilderPattern1(this);
74     }
75 }
```

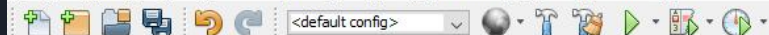
designpattern.BuilderPattern1 > ComputerBuilder > ComputerBuilder >

Output Navigator

60:18 INS

Type here to search

ENG IN 12-11-2019



Start Page x DesignPattern.java x SingletonPattern.java x BuilderPattern1.java x BuilderPattern2.java x

Source History

```
1  * To change this license header, choose License Headers in Project Properties.
2  * To change this template file, choose Tools | Templates
3  * and open the template in the editor.
4  */
5
6  package designpattern;
7
8  /**
9   *
10  * @author shaik
11  */
12
13  public class BuilderPattern2 {
14
15      public static void main(String[] args) {
16          //Using builder to get the object in a single line of code and
17          //without any inconsistent state or arguments management issues
18          BuilderPattern1 comp = new BuilderPattern1.ComputerBuilder(
19              "500 GB", "2 GB").setBluetoothEnabled(true)
20              .setGraphicsCardEnabled(true).build();
21          System.out.println(comp.toString());
22      }
23
24  }
25
```

designpattern.BuilderPattern2

Output Navigator

13:1

INS



Type here to search



ENG

IN

12:38

12-11-2019



Factory Method Pattern

define an interface or abstract class for creating an object but let the subclasses decide which class to instantiate



Factory Method Pattern

define an interface or abstract class for creating an object but let the subclasses decide which class to instantiate

Factory Method Pattern allows the sub-classes to choose the type of objects to create.



Factory Method Pattern

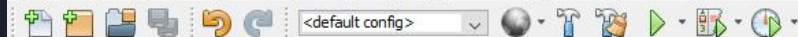
Usage of Factory Design Pattern

- When a class doesn't know what sub-classes will be required to create
- When a class wants that its sub-classes specify the objects to be created.
- When the parent classes choose the creation of objects to its sub-classes.



Factory Method Pattern

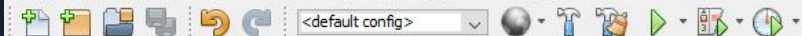
Implementation of Factory Method Pattern Example



...ave SingletonPattern.java x BuilderPattern1.java x BuilderPattern2.java x FactoryDP1.java x FactoryDP2.java x FactoryDP3.java x FactoryDP4.java x FactoryDP5.java x

Source History

```
1  /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package CreationalDesignPattern;
7
8  /**
9   *
10   * @author shaik
11   */
12
13  public abstract class FactoryDP1 { // Computer ABstarct
14
15      public abstract String getRAM();
16      public abstract String getHDD();
17      public abstract String getCPU();
18
19      @Override
20      public String toString(){
21          return "RAM=" +this.getRAM()+" , HDD="+this.getHDD()+" , CPU="+this.getCPU();
22      }
23  }
```



...ava SingletonPattern.java x BuilderPattern1.java x BuilderPattern2.java x FactoryDP1.java x FactoryDP2.java x FactoryDP3.java x FactoryDP4.java x FactoryDP5.java x

Source History

```
14
15 public class FactoryDP2 extends FactoryDP1 { //Personal Computer
16
17     private String ram;
18     private String hdd;
19     private String cpu;
20
21     public FactoryDP2(String ram, String hdd, String cpu){
22         this.ram=ram;
23         this.hdd=hdd;
24         this.cpu=cpu;
25     }
26     @Override
27     public String getRAM() {
28         return this.ram;
29     }
30
31     @Override
32     public String getHDD() {
33         return this.hdd;
34     }
35
36     @Override
37     public String getCPU() {
38         return this.cpu;
39     }
40
41 }
42
```

CreationalDesignPattern.FactoryDP2

Output

15:65

INS

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

SingletonPattern.java BuilderPattern1.java BuilderPattern2.java FactoryDP1.java FactoryDP2.java FactoryDP3.java FactoryDP4.java FactoryDP5.java

Source History

```
11 //
12
13
14 public class FactoryDP3 extends FactoryDP1 { //Server Computer
15
16     private String ram;
17     private String hdd;
18     private String cpu;
19
20     public FactoryDP3(String ram, String hdd, String cpu){
21         this.ram=ram;
22         this.hdd=hdd;
23         this.cpu=cpu;
24     }
25     @Override
26     public String getRAM() {
27         return this.ram;
28     }
29
30     @Override
31     public String getHDD() {
32         return this.hdd;
33     }
34
35     @Override
36     public String getCPU() {
37         return this.cpu;
38     }
39
40 }
41
```

CreationDesignPattern.FactoryDP3

Output

14:62 INS

Type here to search

ENG INTL 13:01 12-11-2019



...ave SingletonPattern.java x BuilderPattern1.java x BuilderPattern2.java x FactoryDP1.java x FactoryDP2.java x FactoryDP3.java x FactoryDP4.java x FactoryDP5.java x

Source History

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package CreationalDesignPattern;
7
8  /**
9   *
10   * @author shaik
11   */
12
13
14
15
16  public class FactoryDP4 { //Computer Factory
17
18      public static FactoryDP1 getComputer(String type, String ram, String hdd, String cpu){
19          if("PC".equalsIgnoreCase(type)) return new FactoryDP2(ram, hdd, cpu);
20          else if("Server".equalsIgnoreCase(type)) return new FactoryDP3(ram, hdd, cpu);
21
22          return null;
23      }
24  }
25
```

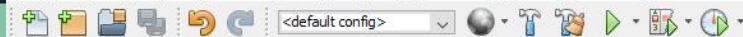
CreationalDesignPattern.FactoryDP4

Output

16:45

INS





...ave SingletonPattern.java x BuilderPattern1.java x BuilderPattern2.java x FactoryDP1.java x FactoryDP2.java x FactoryDP3.java x FactoryDP4.java x FactoryDP5.java x

Source History

```
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package CreationalDesignPattern;
7
8  /**
9   *
10   * @author shaik
11   */
12
13
14  public class FactoryDP5 { //Client
15
16      public static void main(String[] args) {
17          FactoryDP1 pc = FactoryDP4.getComputer("pc","2 GB","500 GB","2.4 GHz");
18          FactoryDP1 server = FactoryDP4.getComputer("server","16 GB","1 TB","2.9 GHz");
19          System.out.println("Factory PC Config: "+pc);
20          System.out.println("Factory Server Config: "+server);
21      }
22
23  }
24
```

CreationalDesignPattern.FactoryDP5

Output



Abstract Factory Pattern

Abstract Factory pattern is almost similar to Factory Pattern except the fact that its more like factory of factories.



Abstract Factory Pattern

define an interface or abstract class for creating families of related (or dependent) objects but without specifying their concrete sub-classes



Abstract Factory Pattern

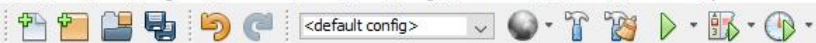
Advantage of Abstract Factory Pattern

- Abstract Factory Pattern isolates the client code from concrete (implementation) classes.
- It eases the exchanging of object families.
- It promotes consistency among objects.



Abstract Factory Pattern

Implementation of Abstract Factory Method Pattern Example



AbstractFP1.java x AbstractFP2.java x AbstractFP3.java x AbstractFP4.java x AbstractFP5.java x AbstractFP6.java x FactoryDP2.java x AbstractFP7.java x AbstractFP8.java x

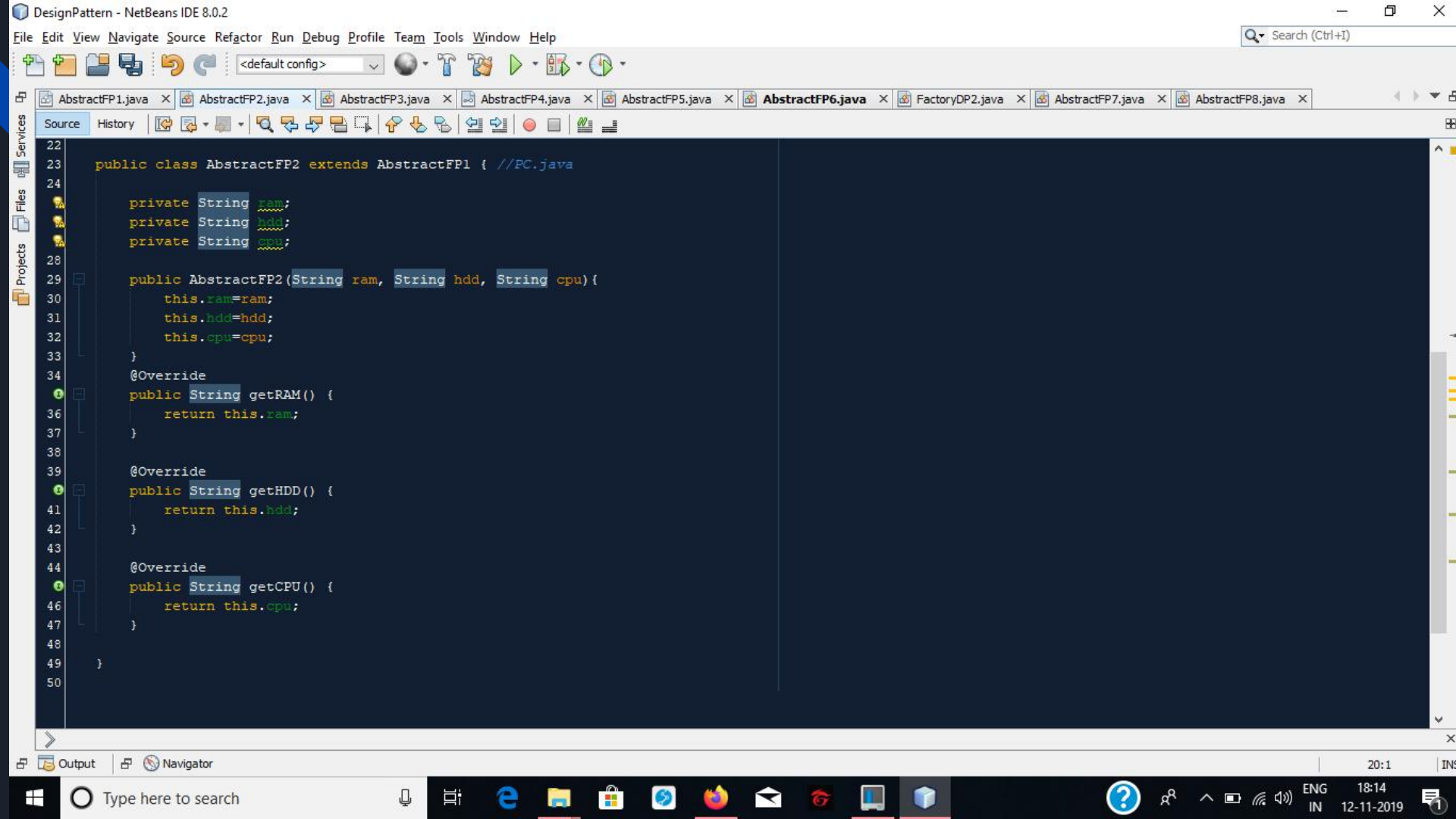
Source History

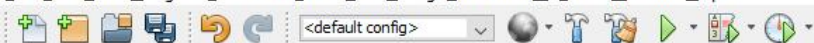
```
7
8  /**
9   *
10  * @author shaik
11  */
12  //1 Computer.java
13  //2 PC.java
14  //3 Server.java
15  //4 ComputerAbstractFactory.java
16  //5 PCFactory.java
17  //6 ServerFactory.java
18  //7 ComputerFactory.java
19  //8 TestDesignPatterns.java
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

Output Navigator

20:1

INS



AbstractFP1.java x AbstractFP2.java x AbstractFP3.java x AbstractFP4.java x AbstractFP5.java x **AbstractFP6.java** x FactoryDP2.java x AbstractFP7.java x AbstractFP8.java x

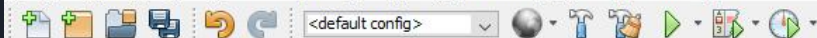
Source History

```
21
22
23 public class AbstractFP3 extends AbstractFP1 {
24
25     private String ram;
26     private String hdd;
27     private String cpu;
28
29     public AbstractFP3(String ram, String hdd, String cpu) {
30         this.ram=ram;
31         this.hdd=hdd;
32         this.cpu=cpu;
33     }
34     @Override
35     public String getRAM() {
36         return this.ram;
37     }
38
39     @Override
40     public String getHDD() {
41         return this.hdd;
42     }
43
44     @Override
45     public String getCPU() {
46         return this.cpu;
47     }
48
49 }
50
```

Output Navigator

20:1

INS



<default config>

AbstractFP1.java x AbstractFP2.java x AbstractFP3.java x AbstractFP4.java x AbstractFP5.java x **AbstractFP6.java** x FactoryDP2.java x AbstractFP7.java x AbstractFP8.java xServices
Files
Projects

```
8  /**
9   *
10  * @author shaik
11  */
12
13  //1 Computer.java
14  //2 PC.java
15  //3 Server.java
16  //4 ComputerAbstractFactory.java
17  //5 PCFactory.java
18  //6 ServerFactory.java
19  //7 ComputerFactory.java
20  //8 TestDesignPatterns.java
21
22  public interface AbstractFP4 {
23
24      public AbstractFP1 createComputer();
25
26  }
27
```

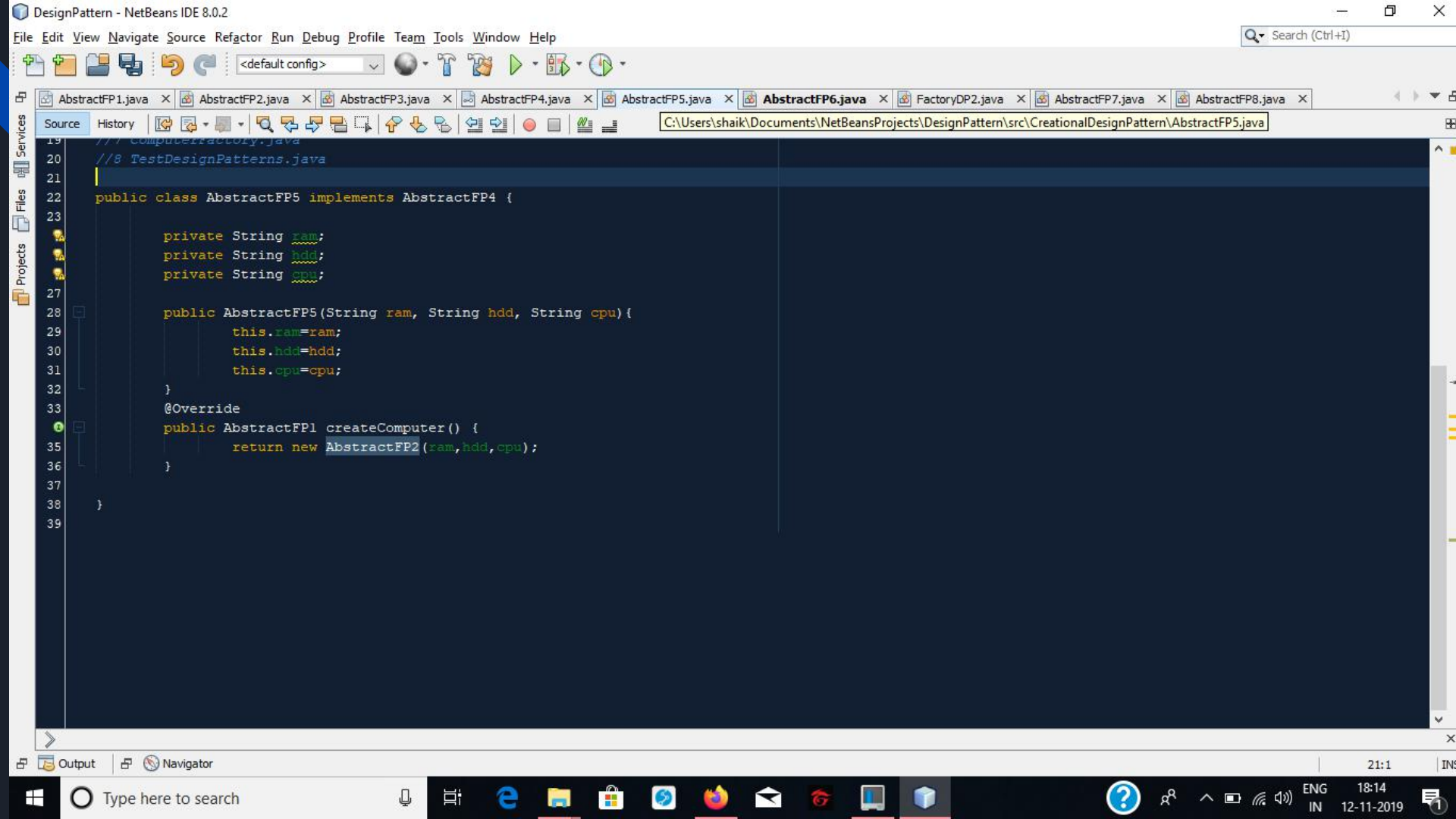
Output Navigator

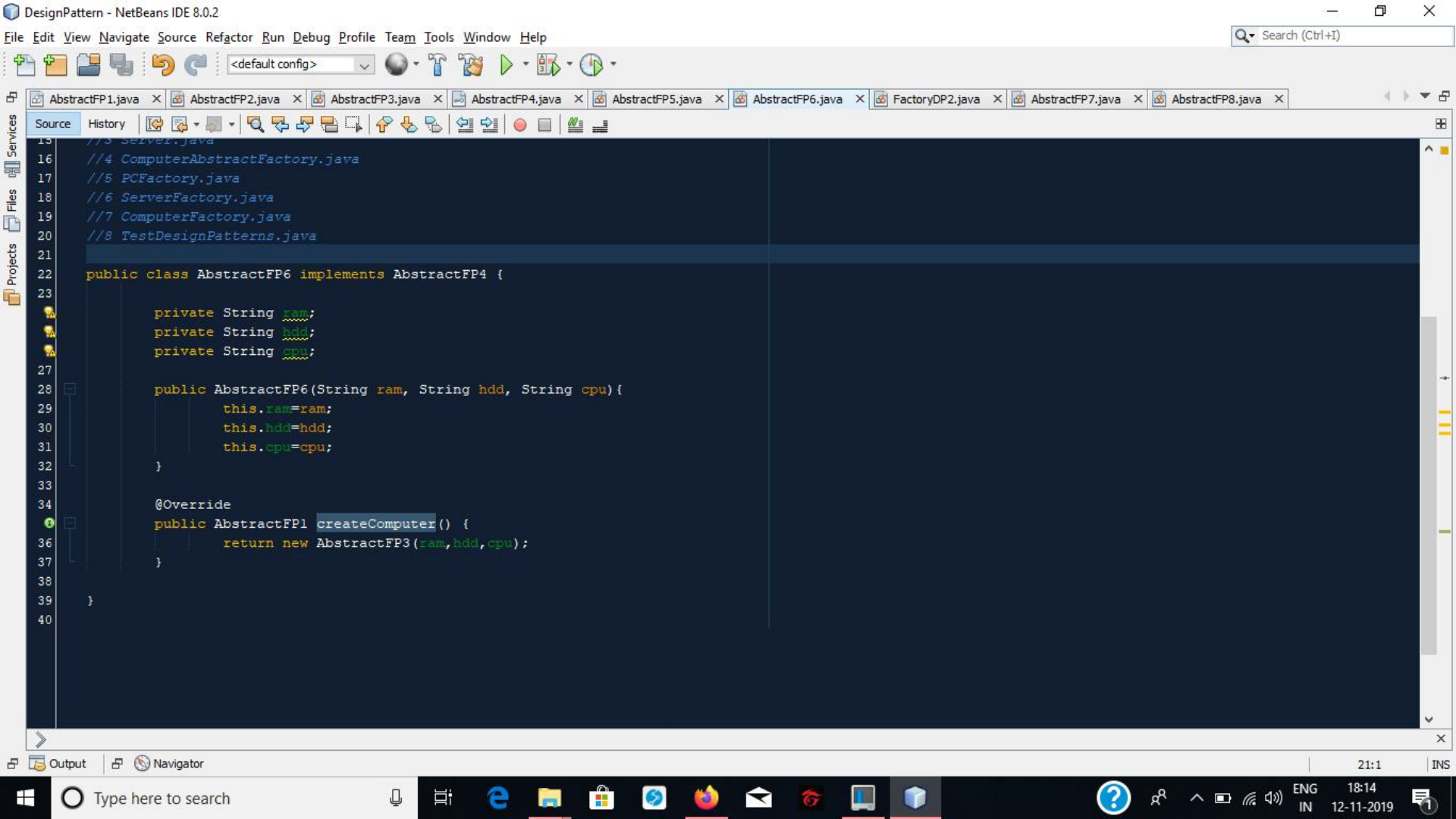
21:1

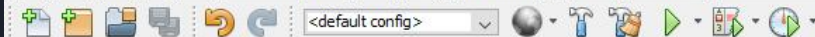


Type here to search

ENG
IN18:14
12-11-2019



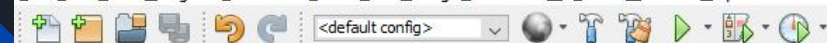




AbstractFP1.java x AbstractFP2.java x AbstractFP3.java x AbstractFP4.java x AbstractFP5.java x AbstractFP6.java x FactoryDP2.java x AbstractFP7.java x AbstractFP8.java x



```
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package CreationalDesignPattern;
7
8  /**
9   *
10   * @author shaik
11   */
12  //1 Computer.java
13  //2 PC.java
14  //3 Server.java
15  //4 ComputerAbstractFactory.java
16  //5 PCFactory.java
17  //6 ServerFactory.java
18  //7 ComputerFactory.java
19  //8 TestDesignPatterns.java
20
21
22  public class AbstractFP7 {
23
24      public static AbstractFP1 getComputer(AbstractFP4 factory){
25          return factory.createComputer();
26      }
27  }
```

AbstractFP1.java x AbstractFP2.java x AbstractFP3.java x AbstractFP4.java x AbstractFP5.java x AbstractFP6.java x FactoryDP2.java x AbstractFP7.java x AbstractFP8.java x

Source History

```
12
13 //1 Computer.java
14 //2 PC.java
15 //3 Server.java
16 //4 ComputerAbstractFactory.java
17 //5 PCFactory.java
18 //6 ServerFactory.java
19 //7 ComputerFactory.java
20 //8 TestDesignPatterns.java
21
22
23 public class AbstractFP8 {
24
25     public static void main(String[] args) {
26         testAbstractFactory();
27     }
28
29     private static void testAbstractFactory() {
30         AbstractFP1 pc = AbstractFP7.getComputer(new AbstractFP5("2 GB", "500 GB", "2.4 GHz"));
31         AbstractFP1 server = AbstractFP7.getComputer(new AbstractFP6("16 GB", "1 TB", "2.9 GHz"));
32         System.out.println("AbstractFactory PC Config::"+pc);
33         System.out.println("AbstractFactory Server Config::"+server);
34     }
35 }
36
37
```

Output Navigator

22:1

IN



Prototype Design Pattern

cloning of an existing object instead of creating new one and can also be customized as per the requirement



Prototype Pattern

The main advantages of prototype pattern are as follows:

- It reduces the need of sub-classing.
- It hides complexities of creating objects.
- The clients can get new objects without knowing which type of object it will be.
- It lets you add or remove objects at runtime.



Prototype Pattern

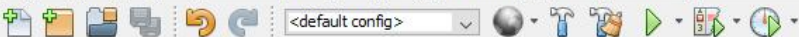
Usage of Prototype Pattern

- When the classes are instantiated at runtime.
- When the cost of creating an object is expensive or complicated.
- When you want to keep the number of classes in an application minimum.
- When the client application needs to be unaware of object creation and representation.



Prototype Pattern

Implementation of Prototype Pattern Example



...ave AbstractFP3.java x AbstractFP4.java x AbstractFP5.java x AbstractFP6.java x FactoryDP2.java x AbstractFP7.java x AbstractFP8.java x EmpProto.java x PrototypePattern.java x

Source

History

```
10  * @author shaik
11  */
12
13
14  import java.util.ArrayList;
15  import java.util.List;
16
17  public class EmpProto implements Cloneable{
18
19      private List<String> empList;
20
21      public EmpProto() {
22          empList = new ArrayList<String>();
23      }
24
25      public EmpProto(List<String> list) {
26          this.empList=list;
27      }
28      public void loadData(){
29          //read all employees from database and put into the list
30          empList.add("Pankaj");
31          empList.add("Raj");
32          empList.add("David");
33          empList.add("Lisa");
34      }
35
36      public List<String> getEmpList() {
37          return empList;
38      }
39  }
```

CreationalDesignPattern.EmpProto

Output

Navigator

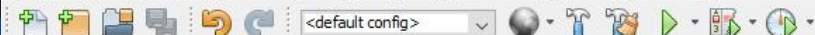
39:1

INS



```
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50
```

```
    empList.add("Pankaj");  
    empList.add("Raj");  
    empList.add("David");  
    empList.add("Lisa");  
}  
  
public List<String> getEmpList() {  
    return empList;  
}  
  
@Override  
public Object clone() throws CloneNotSupportedException {  
    List<String> temp = new ArrayList<String>();  
    for(String s : this.getEmpList()) {  
        temp.add(s);  
    }  
    return new EmpProto(temp);  
}
```




...ava AbstractFP3.java x AbstractFP4.java x AbstractFP5.java x AbstractFP6.java x FactoryDP2.java x AbstractFP7.java x AbstractFP8.java x EmpProto.java x PrototypePattern.java x



```
13
14 import java.util.List;
15
16
17 public class PrototypePattern {
18
19     public static void main(String[] args) throws CloneNotSupportedException {
20         EmpProto emps = new EmpProto();
21         emps.loadData();
22
23         //Use the clone method to get the Employee object
24         EmpProto empsNew = (EmpProto) emps.clone();
25         EmpProto empsNew1 = (EmpProto) emps.clone();
26         List<String> list = empsNew.getEmpList();
27         list.add("John");
28         List<String> list1 = empsNew1.getEmpList();
29         list1.remove("Pankaj");
30
31         System.out.println("emps List: "+emps.getEmpList());
32         System.out.println("empsNew List: "+list);
33         System.out.println("empsNew1 List: "+list1);
34     }
35
36 }
37
```

CreationalDesignPattern.PrototypePattern > main >

Output Navigator



Assignment

1. Create a Builder Pattern for a Mobile which has following parameters
 - a. Ram
 - b. Os
 - c. Internal Storage
 - d. Battery Capacity
 - e. Company
2. Create a Factory Pattern and Abstract Factory Pattern of Vehicles which to create a car, bus, bike, truck which has input parameters as No. of passengers, No of Tyres, Engine Capacity etc
3. Create a Prototype pattern for an educational institute which should have some sample lists like students, faculty, incharge, attenders etc



References

- <https://www.javatpoint.com/singleton-design-pattern-in-java>
- <https://sandeepdass003.wordpress.com/2018/02/23/eager-and-lazy-instantiation-in-singleton-design-pattern-implementation/>
- <https://www.javatpoint.com/builder-design-pattern>
- <https://www.journaldev.com/1425/builder-design-pattern-in-java>
- <https://www.javatpoint.com/factory-method-design-pattern>
- <https://www.journaldev.com/1392/factory-design-pattern-in-java>
- <https://www.journaldev.com/1418/abstract-factory-design-pattern-in-java>
- <https://www.javatpoint.com/abstract-factory-pattern>
- <https://www.javatpoint.com/prototype-design-pattern>
- <https://www.journaldev.com/1440/prototype-design-pattern-in-java>