

**2022-2023**

**Département Mathématiques et Informatique  
GLSID 3 – S5**

**Design Patterns**

Rapport de devoir 1

Travail à Rendre Design patterns

**Préparé par : Fatima Zahra HASBI**

**Encadré par : Mr Mohamed YOUSSEFI**

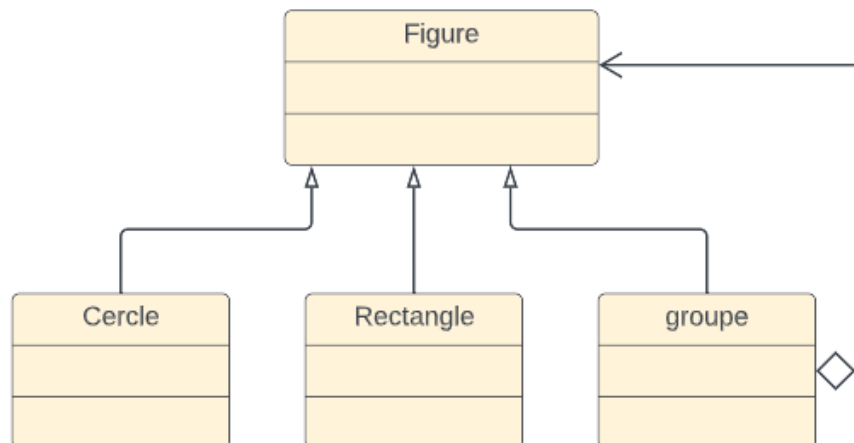
## Design patterns

### Exercice 1 :

Créer les diagrammes de classes en mentionnant les designs patterns appropriés pour les situations suivantes :

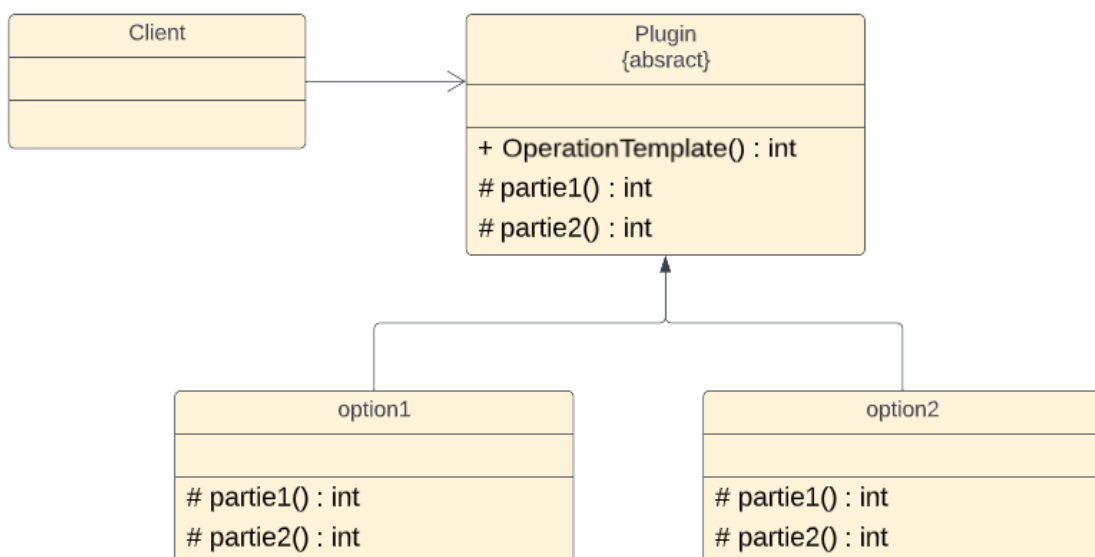
- 1- Une figure peut être soit un cercle, un rectangle ou un groupe de figures.

Composite pattern



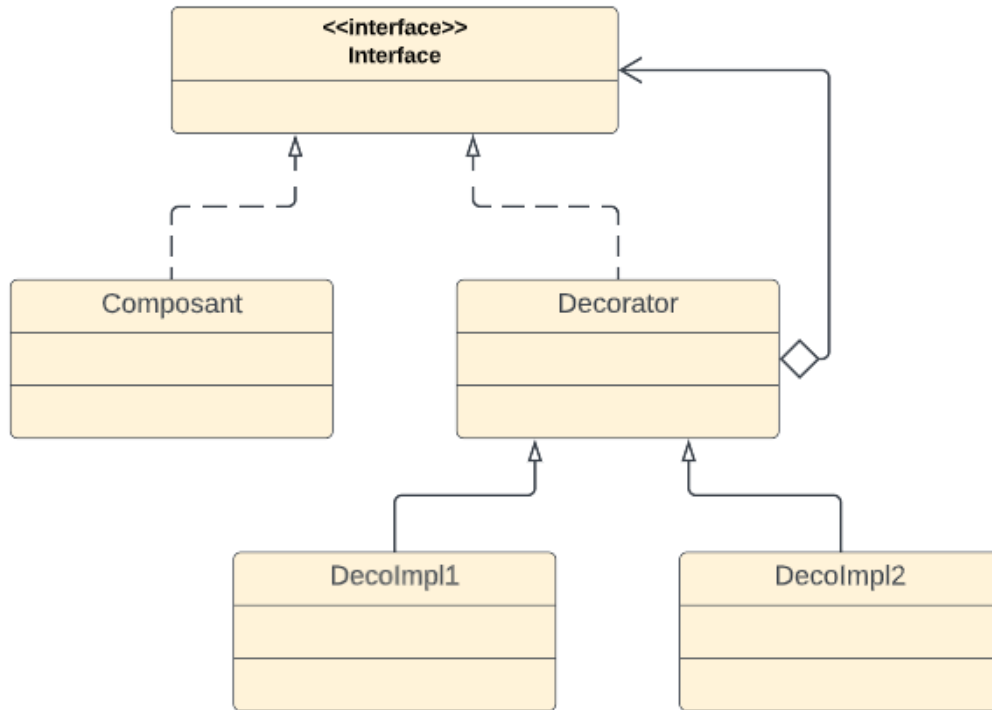
- 2- Un plugin contient une opération implémentant le squelette d'un algorithme dont deux parties (partie1 et partie2) sont variables. On voudrait laisser le développeur la possibilité d'implémenter les deux parties manquantes de cet algorithme et on voudrait aussi que l'application cliente puisse instancier une implémentation concrète du plugin sans connaître sa classe d'implémentation.

Template Pattern



- 3- On dispose d'un **composant** implémentant une interface qui définit une opération « *traitement()* ». On voudrait rattacher à ce composant des responsabilités supplémentaires sans modifier son code source. C'est-à-dire envelopper l'exécution de la méthode *traitement* par d'autres traitements avant et après son exécution.

## Decorator pattern



## Exercice 2 :

On souhaite concevoir et développer un Framework qui permet d'effectuer des traitements sur une image. L'image étant représentée par un tableau d'entiers. Le Framework définit deux opérations principales :

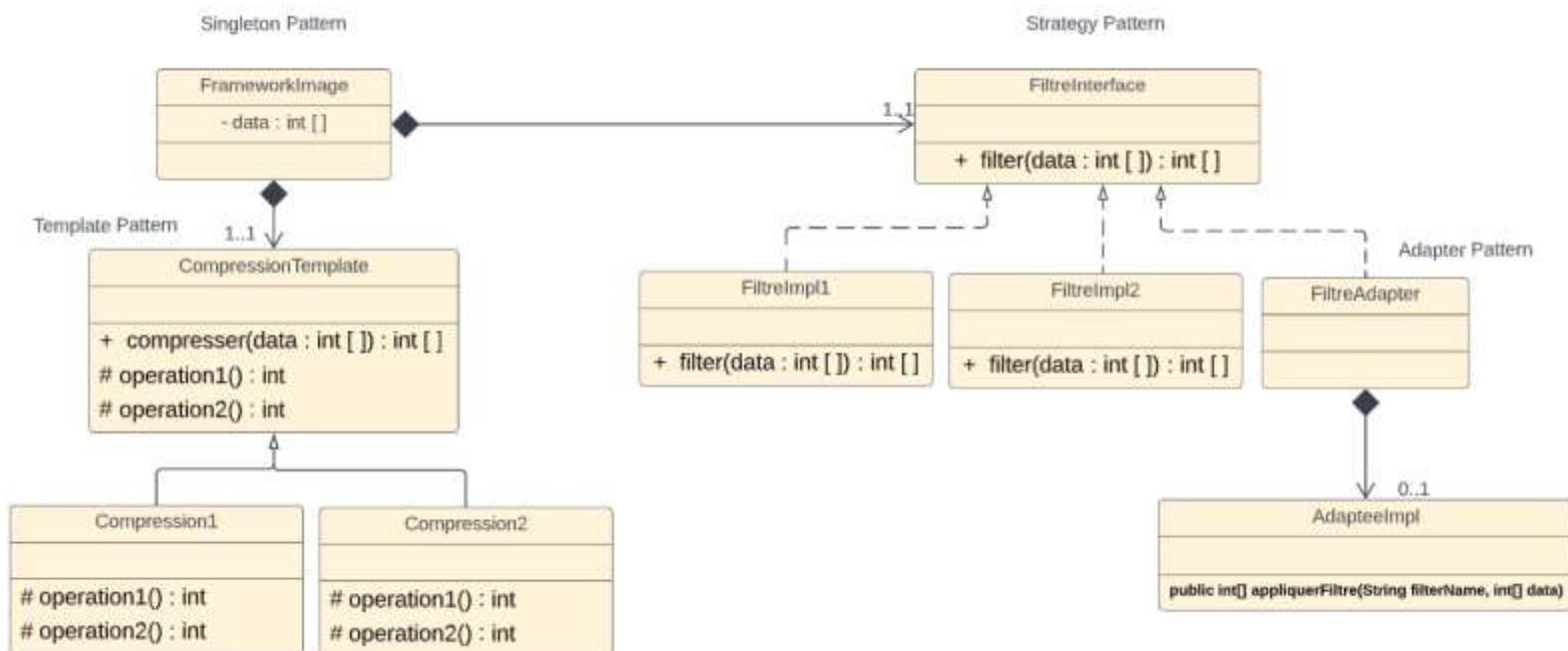
- Une opération qui permet de filtrer l'image dont la signature est :
  - **public int[] filter (int [] data)**
- Une opération qui permet de compresser l'image dont la signature est :
  - **public int[] compresser (int [] data)**

Le Framework doit respecter les critères suivants :

- Il doit être fermé à la modification et ouvert à l'extension.
- L'opération de filtrage à effectuer peut évoluer dans le temps. Cela signifie que l'utilisateur de l'application peut, lui-même, définir de nouvelles implémentations de l'opération de filtrage.
- Au moment de l'exécution, on peut changer dynamiquement la version de l'implémentation de filtrage à appliquer à l'image.
- Permettre au Framework d'utiliser une ancienne implémentation (ImplNonStandard) d'une opération de filtrage dont la signature est :
  - **public int[] appliquerFiltre(String filterName, int[] data)**
- Pour l'opération de compression, on souhaite définir le squelette de l'algorithme de compression et déléguer les détails de cet algorithme aux sous classes.

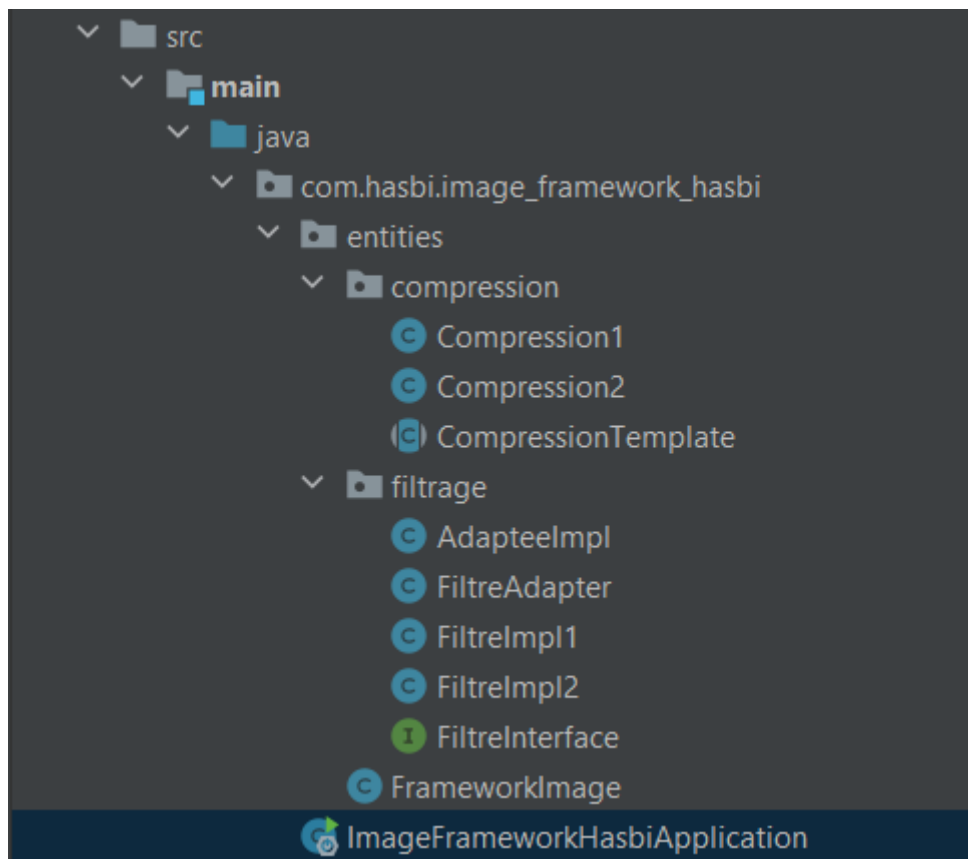
## Questions :

- 1) Etablir un diagramme de classes de ce Framework en appliquant les design patterns appropriés.

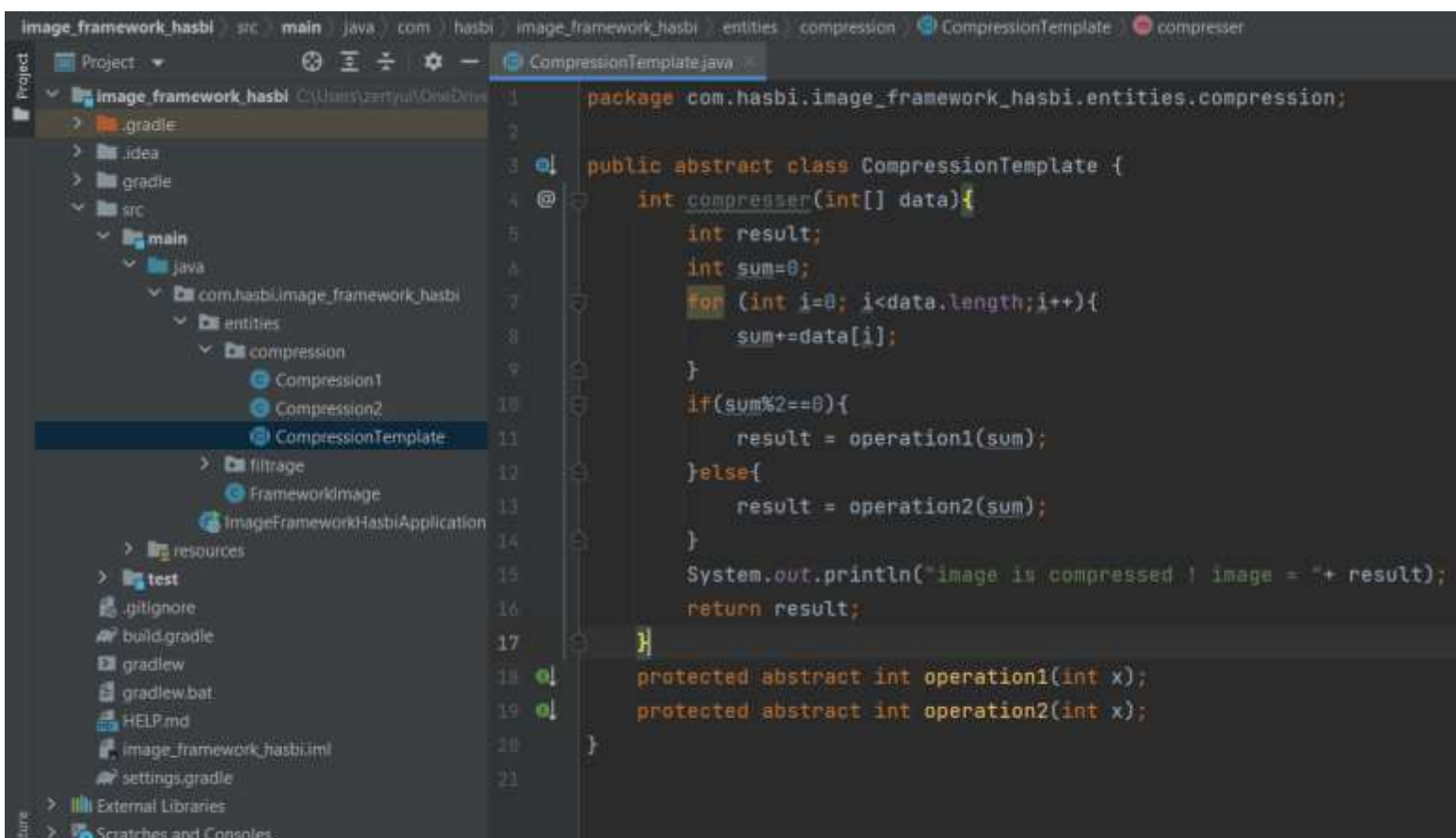


- 2) Ecrire une implémentation Java de ce Framework.

La structure du code :



### Les classes de Compression (Pattern Template)



```

Compression1.java x
1  package com.hasbi.image_framework_hasbi.entities.compression;
2
3  public class Compression1 extends CompressionTemplate {
4      @Override
5      protected int operation1(int x) {
6          return x/2;
7      }
8
9      @Override
10     protected int operation2(int x) { return x; }
13 }

```

```

Compression2.java x
1  package com.hasbi.image_framework_hasbi.entities.compression;
2
3  public class Compression2 extends CompressionTemplate {
4      @Override
5      protected int operation1(int x) {
6          return x;
7      }
8
9      @Override
10     protected int operation2(int x) {
11         return (x/2)+1;
12     }
13 }

```

### Les classes de Filtrage (Pattern Strategy et Pattern Adapter)

```

FiltreInterface.java x
1  package com.hasbi.image_framework_hasbi.entities.filtrage;
2
3  public interface FiltreInterface {
4      public int[] filter(int[] image);
5  }
6
7

```

```
FiltreImpl1.java x
1 package com.hasbi.image_framework_hasbi.entities.filtrage;
2
3 public class FiltreImpl1 implements FiltreInterface {
4     @Override
5     public int[] filter(int[] image) {
6         for (int i = 0; i < image.length; i++) {
7             image[i] += 1;
8         }
9         System.out.println("simple filter 1 is applied!");
10        System.out.println("Image after filter 1: ");
11        System.out.print("[");
12        for (int i = 0; i < image.length; i++) {
13            System.out.print(" " + image[i]);
14        }
15        System.out.print("]");
16        return image;
17    }
18 }
```

```
FiltreImpl2.java x
1 package com.hasbi.image_framework_hasbi.entities.filtrage;
2
3 public class FiltreImpl2 implements FiltreInterface {
4     @Override
5     public int[] filter(int[] image) {
6         for (int i = 0; i < image.length; i++) {
7             image[i] += 2;
8         }
9         System.out.println("simple filter 2 is applied!");
10        System.out.println("Image after filter 2: ");
11        System.out.print("[");
12        for (int i = 0; i < image.length; i++) {
13            System.out.print(" " + image[i]);
14        }
15        System.out.print("]");
16        return image;
17    }
18 }
```



## Pattern Adapter

### Classe de l'ancien filtre

```
AdapteeImpl.java
package com.hasbi.image_framework_hasbi.entities.filtrage;

2
3 public class AdapteeImpl {
4
5     public int[] appliquerFiltre(String filterName, int[] data){
6
7         for (int i = 0; i < data.length; i++) {
8             data[i]*=2;
9         }
10
11         System.out.println("Old filter is applied! the filter name is : " + filterName);
12         System.out.println("Image after filter : ");
13         System.out.print("[");
14         for (int i = 0; i < data.length; i++) {
15             System.out.print(" "+data[i]);
16         }
17         System.out.print("]");
18
19         return data;
20     }
21 }
```

### L'adaptateur

```
FiltreAdapter.java
1 package com.hasbi.image_framework_hasbi.entities.filtrage;
2
3 public class FiltreAdapter implements FiltreInterface {
4     private AdapteeImpl adaptee = new AdapteeImpl();
5     @Override
6     public int[] filter(int[] image) {
7         int[] data = adaptee.appliquerFiltre("Image*2 filter", image);
8         return data;
9     }
10 }
11 }
```



3) Ecrire le code d'une application qui utilise ce Framework.

**Au lieu de donner la main à l'utilisateur de taper le nom de la classe j'ai fais un menu et il suffit de choisir l'implémentation qu'il veut à partir ce menu :**

```
package com.hasbi.image_framework_hasbi;

import com.hasbi.image_framework_hasbi.entities.FrameworkImage;
import com.hasbi.image_framework_hasbi.entities.compression.Compression1;
import com.hasbi.image_framework_hasbi.entities.compression.Compression2;
import com.hasbi.image_framework_hasbi.entities.filtrage.FiltreAdapter;
import com.hasbi.image_framework_hasbi.entities.filtrage.FiltreImpl1;
import com.hasbi.image_framework_hasbi.entities.filtrage.FiltreImpl2;
import com.hasbi.image_framework_hasbi.entities.filtrage.FiltreInterface;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;

@SpringBootApplication
public class ImageFrameworkHasbiApplication {

    public static void main(String[] args) throws IOException {
        SpringApplication.run(ImageFrameworkHasbiApplication.class,
args);

        int[] image = {1, 3, 4, 5};
        FrameworkImage data = new FrameworkImage(image);
        System.out.println("-----Apply a filter and
compress the image-----");
        System.out.print("The image is presented by the following array
: [");
        for (int i = 0; i < data.getImage().length; i++) {
            System.out.print(" " + data.getImage()[i]);
        }
        System.out.println("]");
        System.out.println("\nMenu 1: (Filter)");

        int filterChoice;
        BufferedReader syll=new BufferedReader (new
InputStreamReader(System.in));
        System.out.println("1-Applly filter 1 (FiltreImpl1)");
        System.out.println("2-Applly filter 2 (FiltreImpl2)");
        System.out.println("3-Applly old version of filter
(AdapteeImple)");
        System.out.println("4-Don't apply any filter");
        do {
            System.out.println("\nChoose from menu 1:");
            filterChoice=Integer.parseInt(syll.readLine());
            switch (filterChoice)
            {
                case 1:{
                    data.setFiltreInterface(new FiltreImpl1());
                    data.getFiltreInterface().filter(data.getImage());
                }
                break;
            }
        } while (true);
    }
}
```

```

        case 2: {
            data.setFiltreInterface(new FiltreImpl2());
            data.getFiltreInterface().filter(data.getImage());
        }
        break;
        case 3: {
            data.setFiltreInterface(new FiltreAdapter());
            data.getFiltreInterface().filter(data.getImage());
        }
        break;
    }
}while (filterChoice!=4);

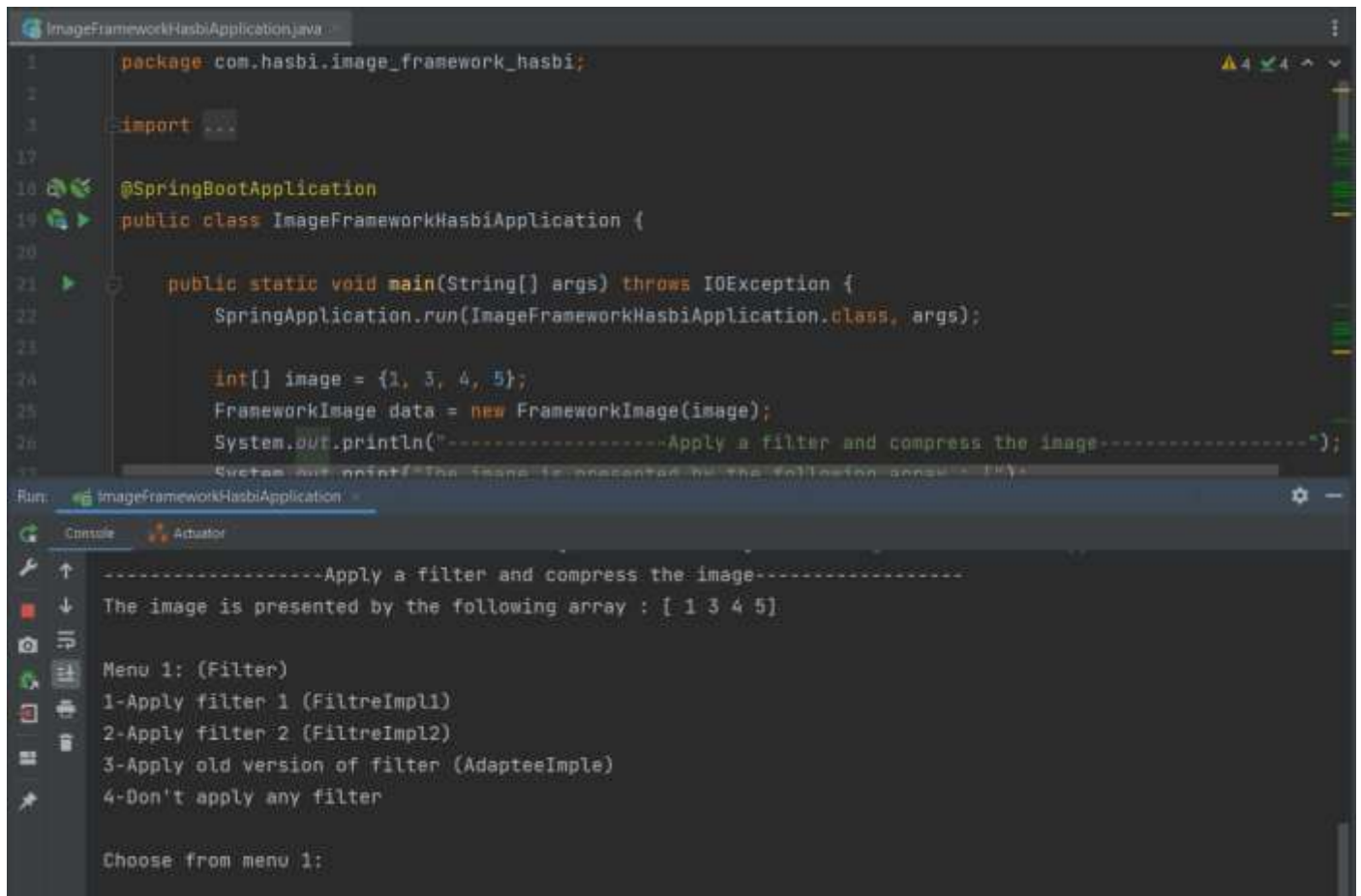
System.out.println("\nMenu 2: (Compression)");

int compressionChoice;
BufferedReader syl2=new BufferedReader (new
InputStreamReader(System.in));
System.out.println("1-Apply Compression 1 ");
System.out.println("2-Apply Compression 2 ");
System.out.println("4-Quit");
do {
    System.out.println("\nChoose from menu 2:");
    compressionChoice=Integer.parseInt(syl2.readLine());
    switch(compressionChoice)
    {
        case 1:{
            data.setCompressionTemplate(new Compression1());
data.getCompressionTemplate().compresser(data.getImage());
            }
            break;
            case 2: {
                data.setCompressionTemplate(new Compression2());
data.getCompressionTemplate().compresser(data.getImage());
            }
            break;
        }
    }while (compressionChoice!=3);

    System.out.println("You quit the frameWork! The program is
finished !");
}
}

```

## Résultat sur console:



The screenshot shows an IDE with a Java file named `ImageFrameworkHasbiApplication.java`. The code is as follows:

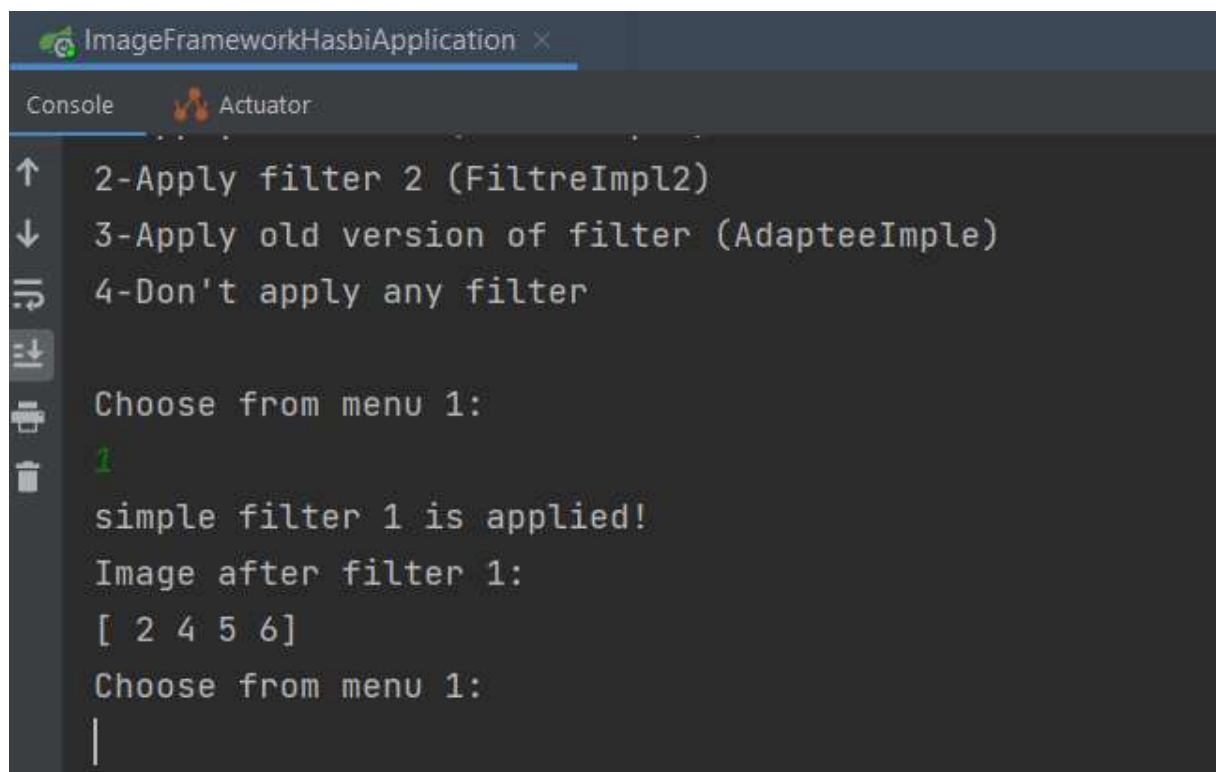
```
1 package com.hasbi.image_framework_hasbi;
2
3 import ...
4
17
18 @SpringBootApplication
19 public class ImageFrameworkHasbiApplication {
20
21     public static void main(String[] args) throws IOException {
22         SpringApplication.run(ImageFrameworkHasbiApplication.class, args);
23
24         int[] image = {1, 3, 4, 5};
25         FrameworkImage data = new FrameworkImage(image);
26         System.out.println("-----Apply a filter and compress the image-----");
27         System.out.printf("The image is presented by the following array : [%s]\n", Arrays.toString(image));
28     }
29 }
```

The console output shows the following sequence of events:

```
-----Apply a filter and compress the image-----
The image is presented by the following array : [ 1 3 4 5]

Menu 1: (Filter)
1-Apply filter 1 (FiltreImpl1)
2-Apply filter 2 (FiltreImpl2)
3-Apply old version of filter (AdapteeImpl)
4-Don't apply any filter

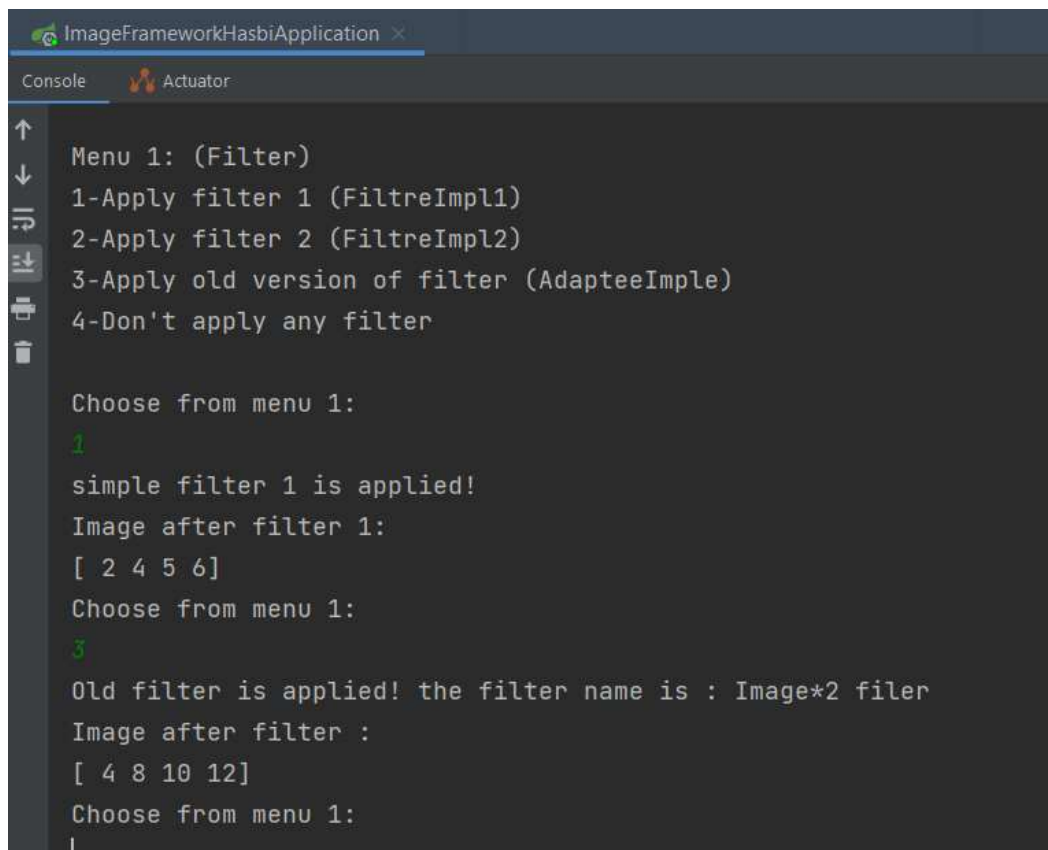
Choose from menu 1:
```



The console continues with the user's input and the application's response:

```
2-Apply filter 2 (FiltreImpl2)
3-Apply old version of filter (AdapteeImpl)
4-Don't apply any filter

Choose from menu 1:
1
simple filter 1 is applied!
Image after filter 1:
[ 2 4 5 6]
Choose from menu 1:
|
```



```
ImageFrameworkHasbiApplication x
Console  🔥 Actuator

↑
↓
⌕
⌕
⌕
⌕

Menu 1: (Filter)
1-Apply filter 1 (FiltreImpl1)
2-Apply filter 2 (FiltreImpl2)
3-Apply old version of filter (AdapteeImple)
4-Don't apply any filter

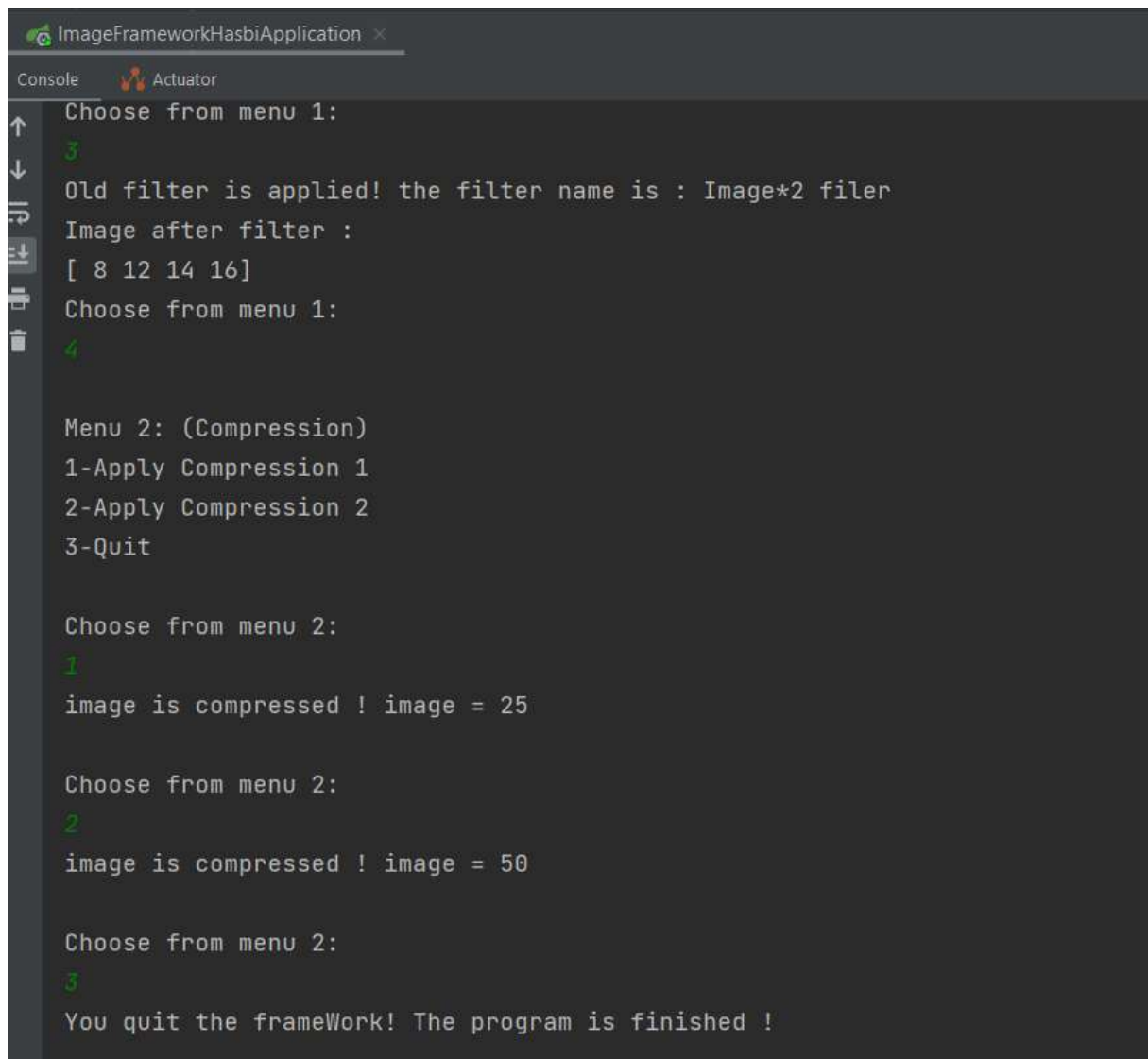
Choose from menu 1:
1
simple filter 1 is applied!
Image after filter 1:
[ 2 4 5 6]
Choose from menu 1:
3
Old filter is applied! the filter name is : Image*2 filer
Image after filter :
[ 4 8 10 12]
Choose from menu 1:
|
```

```
ImageFrameworkHasbiApplication x
Console  Actuator
Menu 1: (Filter)
1-Apply filter 1 (FiltreImpl1)
2-Apply filter 2 (FiltreImpl2)
3-Apply old version of filter (AdapteeImple)
4-Don't apply any filter

Choose from menu 1:
1
simple filter 1 is applied!
Image after filter 1:
[ 2 4 5 6]
Choose from menu 1:
3
Old filter is applied! the filter name is : Image*2 filer
Image after filter :
[ 4 8 10 12]
Choose from menu 1:
4

Menu 2: (Compression)
1-Apply Compression 1
2-Apply Compression 2
4-Quit

Choose from menu 2:
```



The screenshot shows a Java IDE window titled "ImageFrameworkHasbiApplication". The console output is as follows:

```
Choose from menu 1:
3
Old filter is applied! the filter name is : Image*2 filer
Image after filter :
[ 8 12 14 16]
Choose from menu 1:
4

Menu 2: (Compression)
1-Apply Compression 1
2-Apply Compression 2
3-Quit

Choose from menu 2:
1
image is compressed ! image = 25

Choose from menu 2:
2
image is compressed ! image = 50

Choose from menu 2:
3
You quit the frameWork! The program is finished !
```

**Merci!**