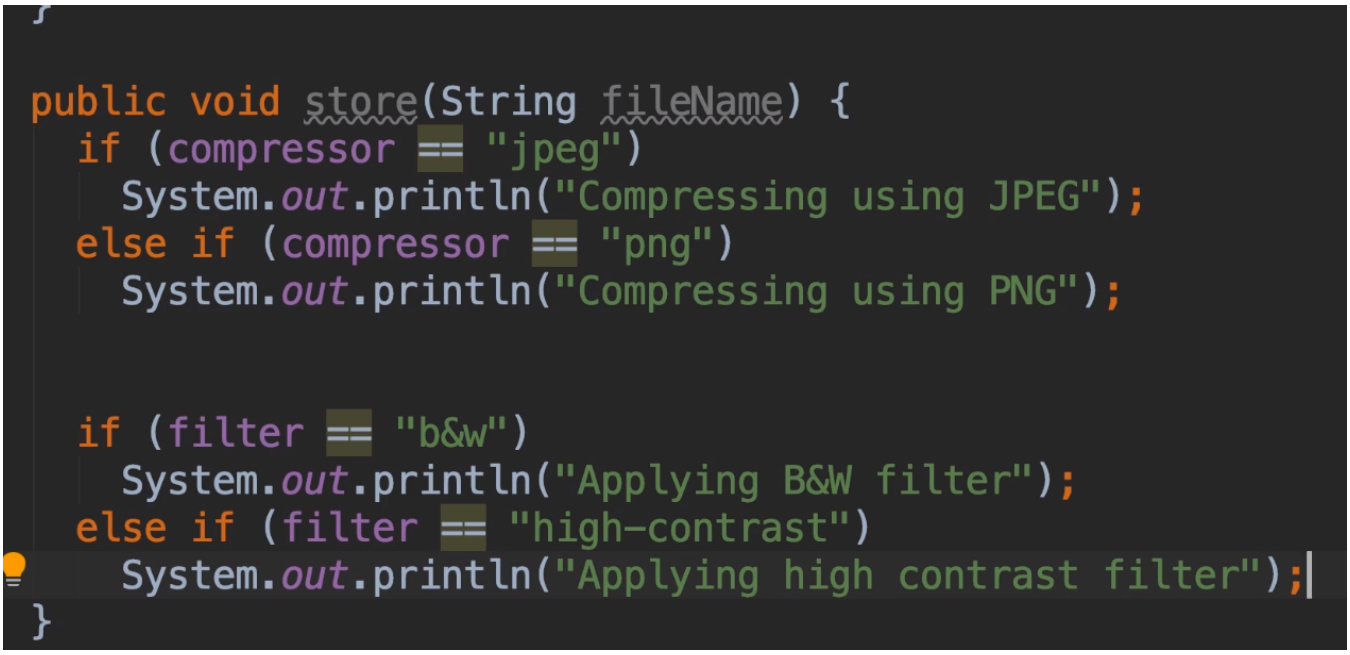
Strategy Pattern

Example Problem

An Application stores images that user upload in the ImageStorage class. The store method takes a file and compresses it using the compressor/file type specified in the compressor instance variable and it then gives the photo a filter e.g. high contrast or B&W which is specified in the filter instance variable :

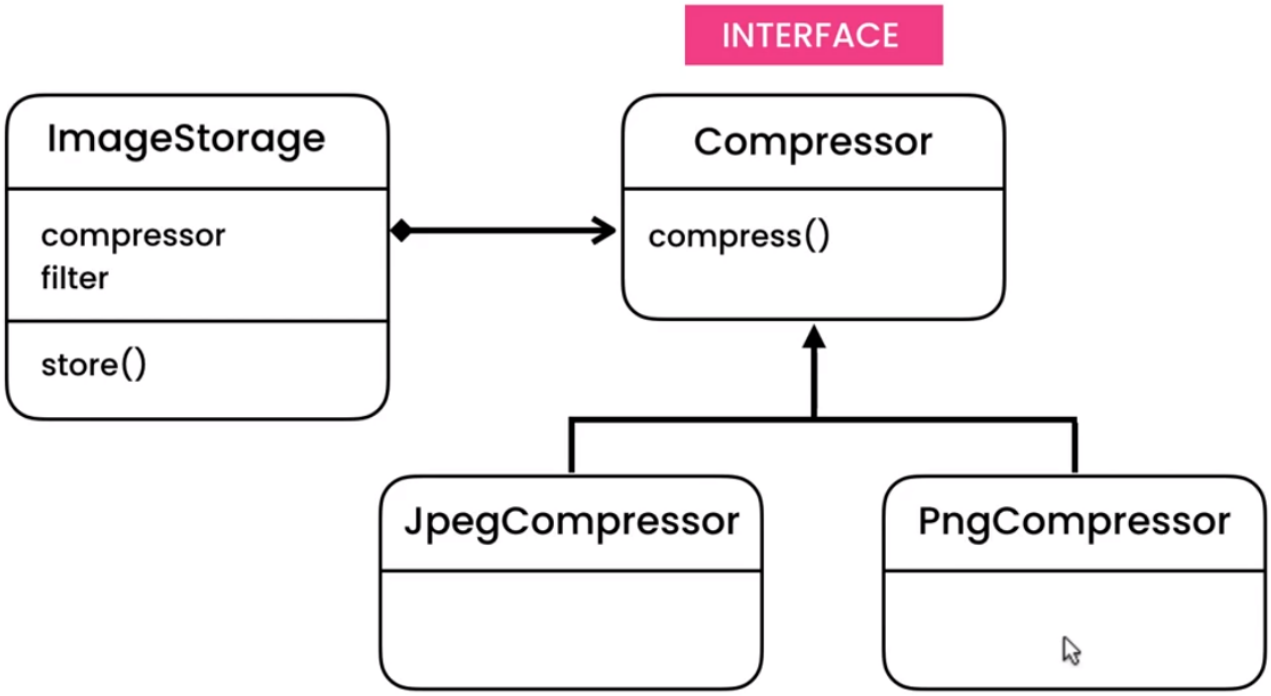


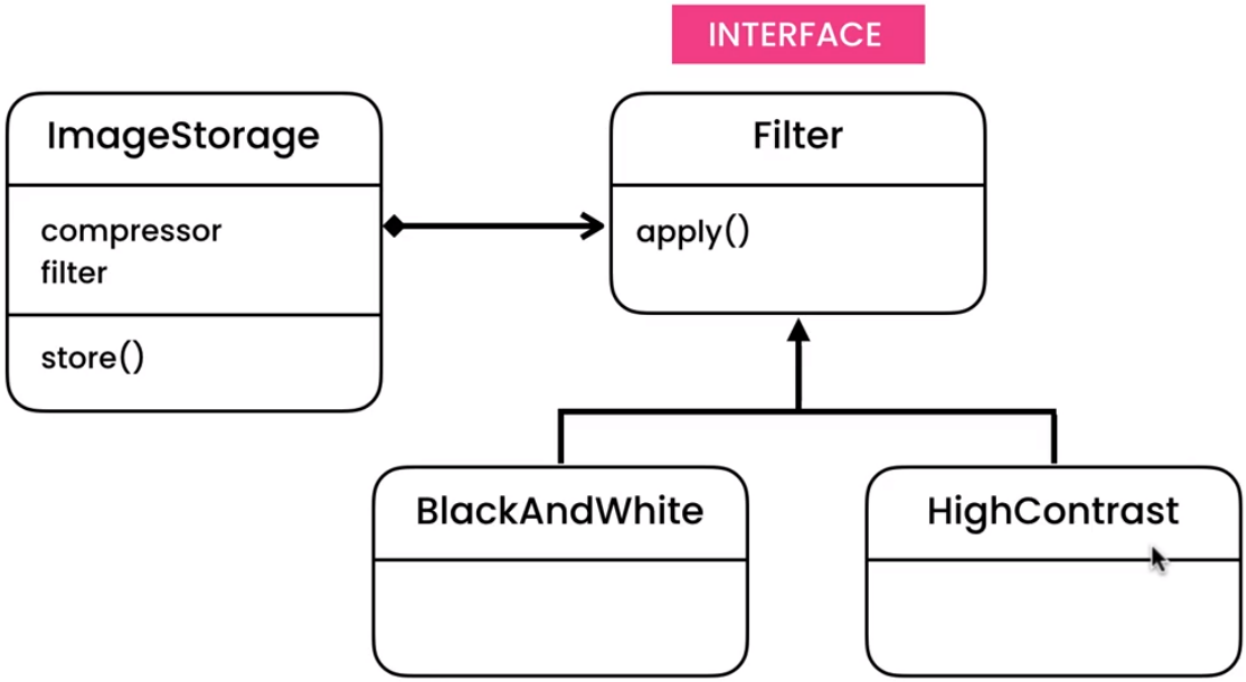
```
}

public void store(String fileName) {
  if (compressor == "jpeg")
    System.out.println("Compressing using JPEG");
  else if (compressor == "png")
    System.out.println("Compressing using PNG");


  if (filter == "b&w")
    System.out.println("Applying B&W filter");
  else if (filter == "high-contrast")
    System.out.println("Applying high contrast filter");
}
```

As can be seen the single responsibility principal and the open closed principal are violated. To solve this we can apply polymorphism to have the image store method behave differently depending on the type of filter or compressor we use, the Strategy pattern enables this.

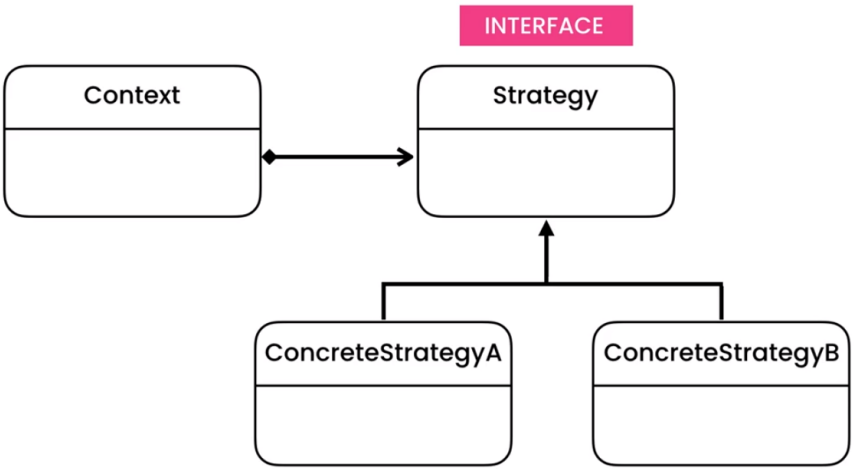
Solution

The compressor and filter functionality instance variables which were previously strings or enums now extend an interface to become Compressor and filter types which then have concrete implementations for each type rather than the if else logic above.

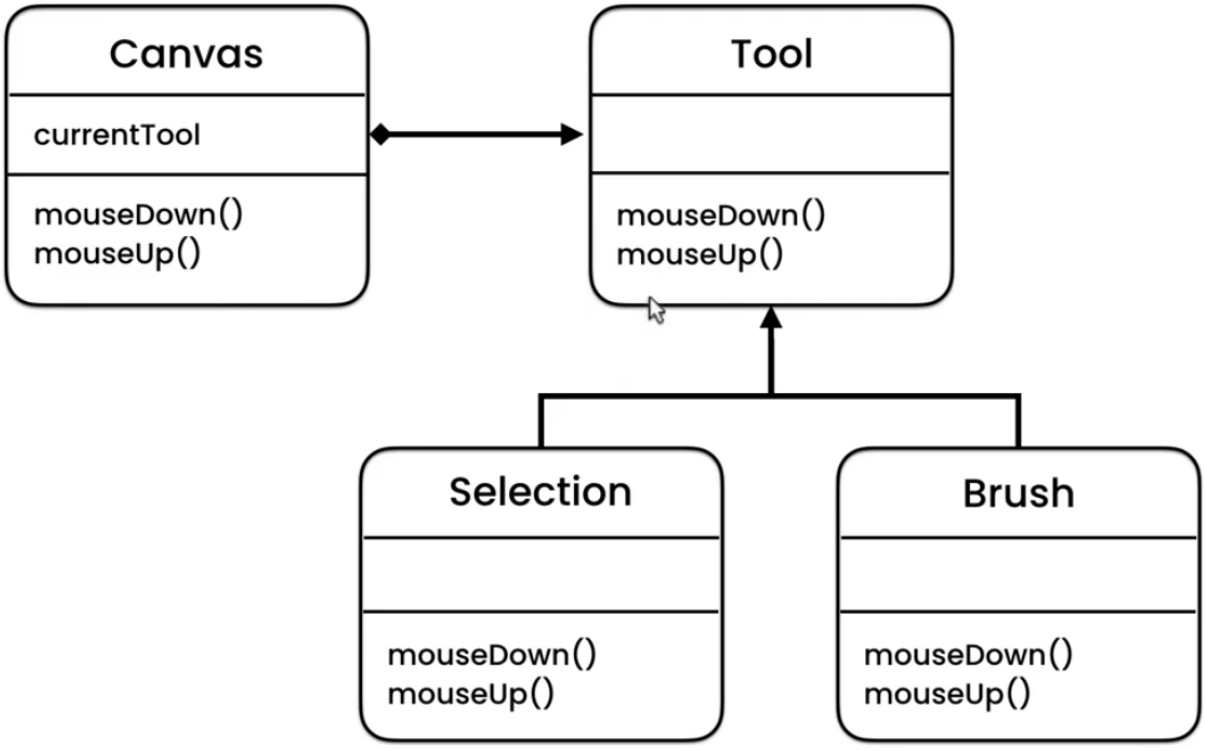




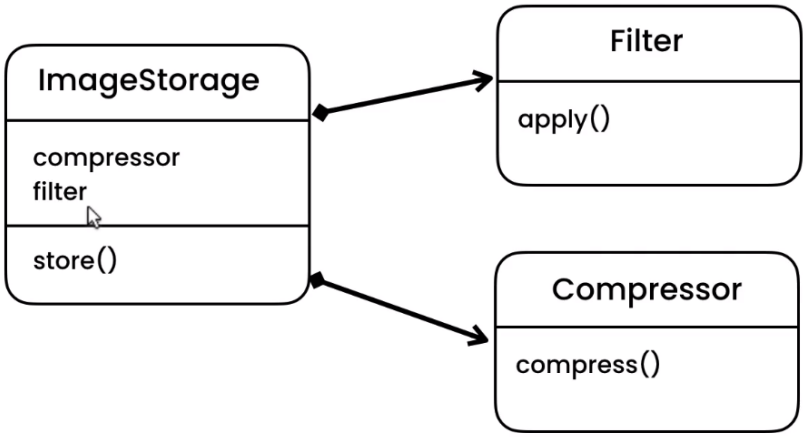
OCP is satisfed as if we need to inroduce new functionality - fiters or compressors then we extend the relevant interface. As per the GOF the strategey interface represents an algorithm to be run in the context class



The Strategy pattern is similar to the state pattern - the difference being that the state pattern focuses on the state of one instance variable (the context class can only have one instance variable / algorithm. Different behaviours occur dependent on the state of this instance vaiable in the strategy pattern :



In contrast in the Strategy pattern we don't have a single state and different behaviours are represented using different strategy objects :



Implementation of the Strategy Pattern

```
public class ImageStorage {
  public void store(String fileName, Compressor compressor, Filter filter) {
    compressor.compress(fileName);

 filter.apply(fileName);
 }
}
```

```
public interface Compressor {
// byte[] compress(byte[] image);
 void compress(String fileName);
}
```

```
public class JpegCompressor implements Compressor {
  @Override
  public void compress (String fileName) {
    System.out.println("Compressing using JPEG");
 }
}
```

```
public class PngCompressor implements Compressor {
  @Override
  public void compress (String fileName) {
    System.out.println("Compressing using PNG");
 }
}
```

```
public interface Filter {
  void apply (String fileName);
}
```

```
public class BlackAndWhiteFilter implements Filter {
  @Override
  public void apply (String fileName) {
    System.out.println("Applying B&W filter");
 }
}
```

```
public class Main {

public static void main(String[] args) {

var imageStorage = new ImageStorage(new JpegCompressor(), new BlackAndWhiteFilter());

//applys the JPEG compressor and then the B&W filter

imageStorage.store("fileName");



//the store method can be updated to take filters and compressors which results in one ImageStorage object

//being capable of applying different filters and compressors on images without the need for a new object a below

var imageStorage = new ImageStorage();

imageStorage.store("fileName", new JpegCompressor(), new BlackAndWhiteFilter());

imageStorage.store("fileName", new PngCompressor(), new BlackAndWhiteFilter());

}

}
```