Writing your own interface

Playable files should be playable!

A use case for a new interface

Let’s pretend that we want to write a media player.

What files should it be able to play?

Audio files and video files are a good start.

If AudioFile extends File and VideoFile also extends File, should the player simply accept a File object?

No, since then we might try to play a TextFile (if we have such a class also extending File). We must use a different strategy.

Playable interface

What if we instead write a new interface, Playable, which declares one single abstract method `play()`?

Then we can let VideoFile and AudioFile both be Files but also be Playable.

So we'd need to write them like this:

```
public class VideoFile extends File implements Playable{...}

public class AudioFile extends File implements Playable{...}
```

And they both need to implement the `play()` method, of course.

The base class File

```
public class File{

  private String name;

  public File(String name){
    this.name = name;
  }

  public String name(){
    return name;
  }
}
```

A simulated play() method

For simplicity, we'll implement the play() method in both subclasses (VideoFile and AudioFile), so that it simply prints a message about what is being played to the standard out.

The play() method

The play() method for AudioFile will print the file's name and type. The type can be music or speech. The types are available as constants in the class.

```
  public void play(){
    System.out.println("Playing the file " + name() +
                       " which is a " +
                       (type==MUSIC?"music":"speech")+
                       " file");
  }

// name() is inherited from File
```

The play() method in VideoFile works similarly

The play() method in the VideoFile works similarly. It simply prints to standard out the file’s name and type of videofile.

Testing the interface

Now, in order to test the interface we can create a heterogenous array of Playable reference, where the references reference a mix of VideoFile and AudioFile objects:

```
Playable[] files={
  new AudioFile("Space oddity.mp3", AudioFile.MUSIC),
  new AudioFile("Cortez the killer.ogg", AudioFile.MUSIC),
  new AudioFile("Dagens eko.mp3", AudioFile.SPEECH),
  new VideoFile("Twin peaks.mp4", VideoFile.TV),
  new VideoFile("The shining.mkv", VideoFile.MOVIE)
};
```

Testing the media player

We should now be able to loop through each element of the array and call a media player with the element as an argument.

The media player could have one method called `playFile` which takes a reference to a `Playable` object as the single argument.

The `playFile` method could then call `play()` on the argument, since every `Playable` object is guaranteed to have such a method. The `play()` method is declared in the interface `Playable`, so each class implementing the interface must also have such a method as a concrete method.

Stupid media player simulation

```
static void playFile(Playable p){
  p.play();
}
```

The full test class

```
public class TestFiles{
  public static void main(String[] args){
    Playable[] files={
      new AudioFile("Space oddity.mp3", AudioFile.MUSIC),
      new AudioFile("Cortez the killer.ogg", AudioFile.MUSIC),
      new AudioFile("Dagens eko.mp3", AudioFile.SPEECH),
      new VideoFile("Twin peaks.mp4", VideoFile.TV),
      new VideoFile("The shining.mkv", VideoFile.MOVIE)
    };
    for(Playable file : files){
      playFile(file);
    }

  }
  static void playFile(Playable p){
    p.play();
  }
}
```

Test run

```
$ javac TestFiles.java && java TestFiles
Playing the file Space oddity.mp3 which is a music file
Playing the file Cortez the killer.ogg which is a music file
Playing the file Dagens eko.mp3 which is a speech file
Playing the file Twin peaks.mp4 which is a TV video file
Playing the file The shining.mkv which is a Movie video file
```

What was the gain from all this?

We created an interface Playable with a play() method, so that we could create some subclasses to File which are also Playable.

This abstraction allowed us to have an array with different types of Playable objects (as references).

This abstraction also allowed us to write a simple media player with a `playFile(Playable p)` method. The team writing the media player, can focus on the simple abstraction that every Playable object has a play() method that we simply can call. How they actually implement play() is up to the authors of the actual classes which implement the Playable interface.

Appendix: the File class

```
public class File{

  private String name;

  public File(String name){
    this.name = name;
  }

  public String name(){
    return name;
  }
}
```

Appendix: the Playable interface

```
public interface Playable{
  public void play();
}
```

Appendix: the AudioFile class

```
public class AudioFile extends File implements Playable{

  public static final int MUSIC = 0;
  public static final int SPEECH= 1;
  public int type;

  public AudioFile(String name, int type){
    super(name);
    this.type=type;
  }
  public int type(){
    return type;
  }
  @Override
  public void play(){
    System.out.println("Playing the file " + name() +
                       " which is a " +
                       (type==MUSIC?"music":"speech")+
                       " file");
  }
}
```

Appendix: the VideoFile class

```
public class VideoFile extends File implements
Playable{
  public static final int TV    = 0;
  public static final int MOVIE = 1;
  public static final int MUSIC = 2;
  public int type;

  public VideoFile(String name, int type){
    super(name);
    this.type=type;
  }
  public int type(){
    return type;
  }
}

// continuous to the right...
```

```
public void play(){
  String genre;
  switch(type){
  case TV:
    genre="TV";
    break;
  case MOVIE:
    genre="Movie";
    break;
  case MUSIC:
    genre="Music";
  default:
    genre="Unknown";
  }
  System.out.println("Playing the file " +
                     name() +
                     " which is a " +
                     genre +
                     " video file");
}
```

Appendix: the TestFiles class

```
public class TestFiles{
  public static void main(String[] args){
    Playable[] files={
      new AudioFile("Space oddity.mp3", AudioFile.MUSIC),
      new AudioFile("Cortez the killer.ogg", AudioFile.MUSIC),
      new AudioFile("Dagens eko.mp3", AudioFile.SPEECH),
      new VideoFile("Twin peaks.mp4", VideoFile.TV),
      new VideoFile("The shining.mkv", VideoFile.MOVIE)
    };
    for(Playable file : files){
      playFile(file);
    }

  }
  // Stupid simulation of a media player
  // capable of playing any Playable file
  // like AudioFile and MusicFile
  static void playFile(Playable p){
    p.play();
  }
}
```