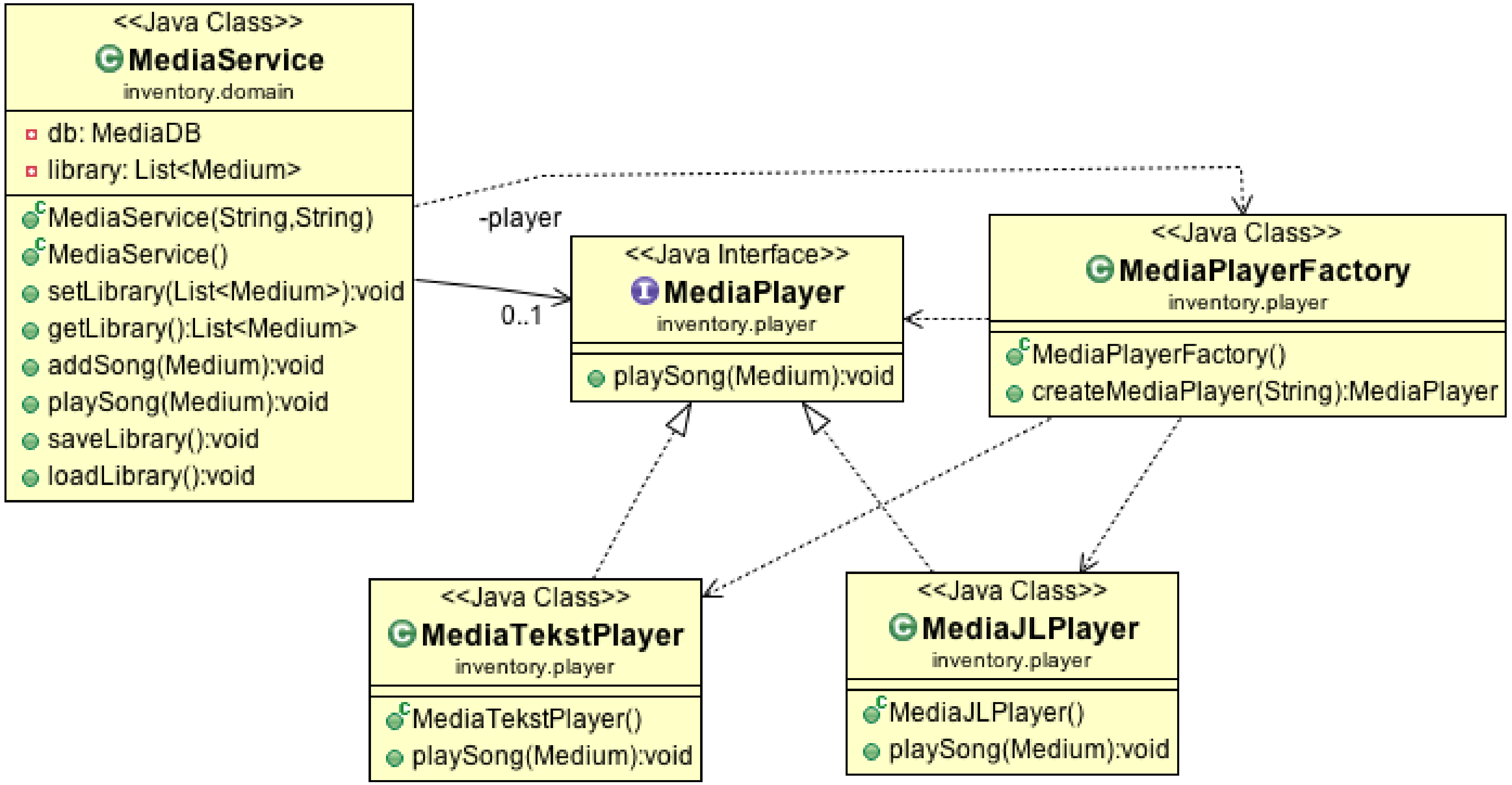
Class

Playing music...



OCP?

```
public class MediaPlayerFactory {

    public MediaPlayer createMediaPlayer(String type){
        MediaPlayer mediaPlayer = null;
        if(type != null && type.equals("text")){
            mediaPlayer = new MediaTekstPlayer();
        } else if(type != null && type.equals("jlplayer")){
            mediaPlayer = new MediaJLPlayer();
        }
        return mediaPlayer;
    }

}
```

Reflection

Is the ability of a computer program …

- to *examine*
- and *modify* the structure and behavior
- of the program at *runtime*

Example

- Class Class

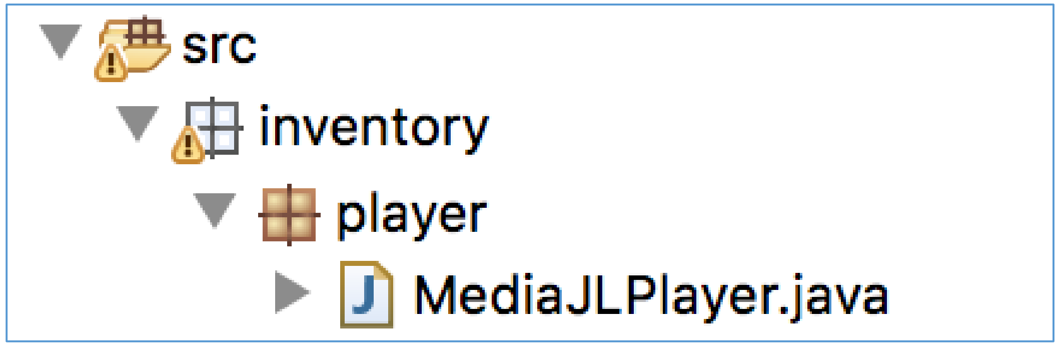
- https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html

Class

```
String playerType = “MediaJLPlayer”;
...


Class dbClass = Class.forName(playerType);

Object dbObject = dbClass.newInstance();

MediaPlayer player = (MediaPlayer) dbObject;
```

MediaJLPlayer

Watch out!


src
inventory
player
MediaJLPlayer.java

```
String playerFullName = "inventory.player." + playerType;

Class dbClass = Class.forName(playerFullName );
```

Fully qualified name !

Watch out!

```
String dbType = “MediaFileDb”;
Class dbClass = Class.forName(dbType);
Object dbObject = dbClass.newInstance();
```

Uses default constructor!

```
public class MediaSQLDB implements MediaDB {
    public MediaSQLDB () {…}
    …
```

```
public class MediaFileDb implements MediaDB {
    public MediaFileDb(String fileName) {…}
    …
```

✗

Solution: init method

```
public class MediaFileDb implements MediaDb {
		private String fileName = ”media.txt";

		public MediaFileDb() {
		}

		public void init(String fileName) {
	setFileName(fileName);
		}

		...
}
```

Call this after object is created

Solution: reflection again

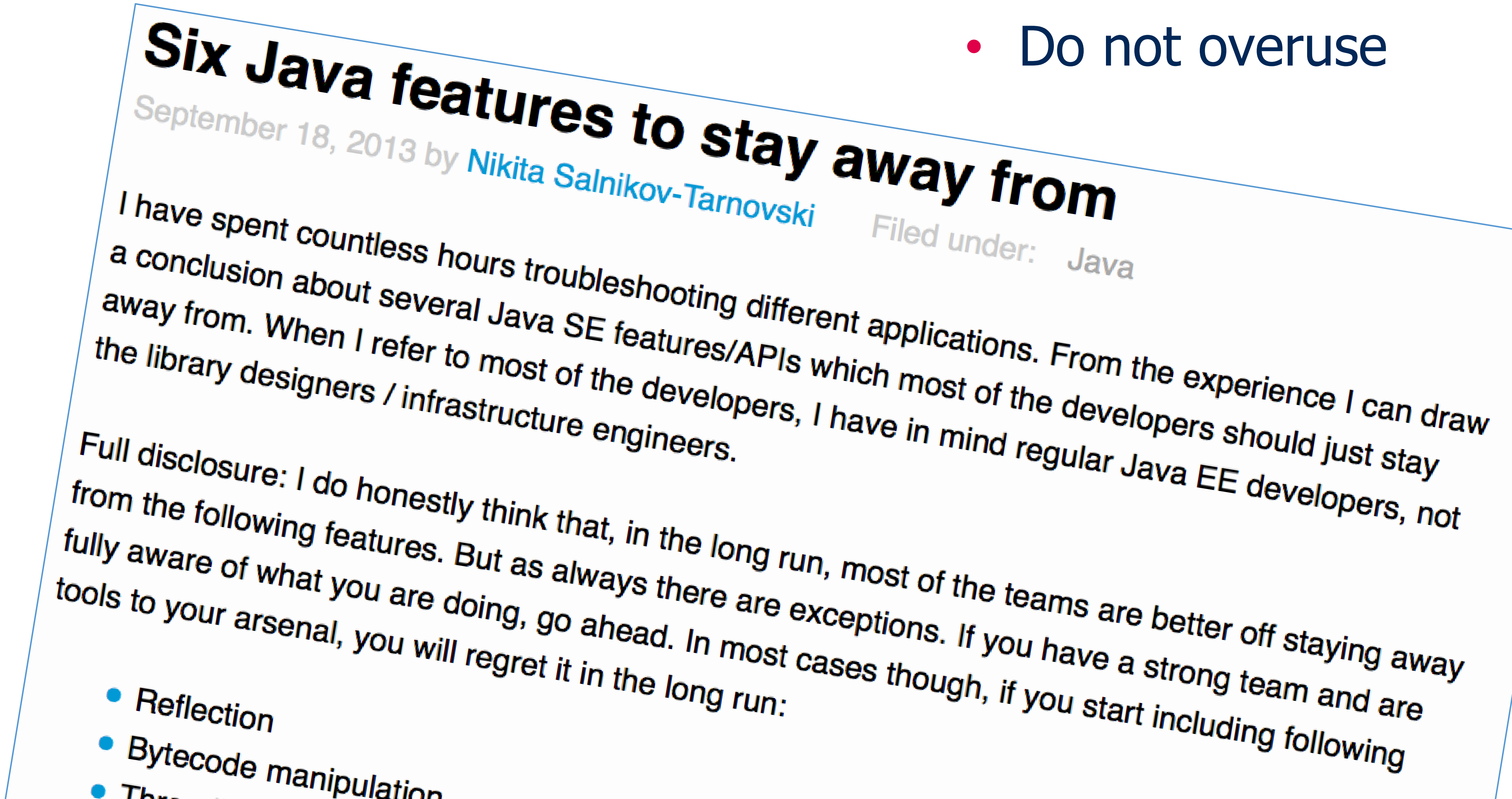
```
Class<?> theClass = Class.forName("com.foo.MyClass");

Constructor<?> constructor =
            theClass.getConstructor(String.class, Integer.class);

Object instance = constructor.newInstance("stringparam", 42);
```

Look for constructor with 2 parameters: String and int

Use this constructor

BUT

- Be careful
- Do not overuse



Six Java features to stay away from

September 18, 2013 by Nikita Salnikov-Tarnovski Filed under: Java

I have spent countless hours troubleshooting different applications. From the experience I can draw a conclusion about several Java SE features/APIs which most of the developers should just stay away from. When I refer to most of the developers, I have in mind regular Java EE developers, not the library designers / infrastructure engineers.

Full disclosure: I do honestly think that, in the long run, most of the teams are better off staying away from the following features. But as always there are exceptions. If you have a strong team and are fully aware of what you are doing, go ahead. In most cases though, if you start including following tools to your arsenal, you will regret it in the long run:

- Reflection
- Bytecode manipulation
- Thr

?

Java
has class