-ER

Alternatives, Clients, MVC

Yegor Bugayenko

Lecture #5 out of 8
90 minutes

"When you need a manager, it's often a sign that the managed are just plain old data structures and that the manager is the smart procedure doing the real work"

— Carlo Pescio
Your Coding Conventions Are Hurting You, 2011

Chapter #1:

Examples and Alternatives

Parser

```
1  class Parser {
2    static int parseInt(String t) {
3      // Parse String into Integer
4    }
5    static float parseFloat(String t) {
6      // Parse String into Float
7    }
8    // And many more methods...
9  }
10
11 int x = Parser.parseInt("42");
```

```
1  class StringAsInt implements Number {
2    private final String txt;
3    StringAsInt(String t) { this.txt = t; }
4    @Override int intValue() {
5      // Parse String into Integer
6      // and return the value
7    }
8  }
9
10 Number n = new StringAsInt("42");
11 int x = n.intValue();
```

Reader

```
1  class Reader {
2    static char readChar(InputStream i) {
3      // Read the next char from the
4      // stream and return it, or NULL
5      // if the stream is at the EOF
6    }
7  }
8
9  InputStream i = new FileInputStream(..);
10 char c = Reader.readChar(i);
```

```
1  class Chars
2    private final InputStream is;
3    Chars(InputStream i)
4      this.is = i;
5    char next()
6      // Read the next char from the
7      // stream and throw exception
8      // if !exists()
9    bool exists()
10     // Return TRUE if not EOF
11
12 InputStream i = new FileInputStream(..);
13 Chars chars = new Chars(i);
14 char c = chars.next();
```

Controller

```
1  class SimpleController {
2    @GET
3    @Path("/index")
4    HttpResponse index(HttpRequest e) {
5      // Build an index page and return
6    }
7    @POST
8    @Path("/update")
9    HttpResponse update(HttpRequest e) {
10     // Save new user information
11     // and return HTTP 303
12   }
13 }
```

```
1  class IndexPage implements HttpPage
2    HttpResponse process(HttpRequest e) {
3      // Build an index page and return
4    }
5  class UpdatePage implements HttpPage
6    HttpResponse process(HttpRequest e) {
7      // Save new user information
8      // and return HTTP 303
9    }
10
11 new AllPages(
12   new IndexPage(),
13   new UpdatePage()
14 );
```

Validator

```
1  class Validator {
2    bool isValid(int age) {
3      return age >= 18;
4    }
5  }
6  int a = 23;
7  Validator v = new Validator();
8  if (!v.isValid(a)) {
9    throw new Exception(
10     "Age is not valid"
11   );
12 }
```

```
1  interface Age
2    int value();
3  class DefaultAge implements Age
4    private final int a;
5    DefaultAge(int a)
6      this.a = a;
7    @Override int value()
8      return this.a;
9  class ValidAge implements Age {
10   private final Age origin;
11   ValidAge(Age age)
12     this.origin = age;
13   @Override int value()
14     int v = this.origin.value();
15     if (v < 18)
16       throw new Exception("Age is not valid");
17     return v;
18
19 Age a = new ValidAge(new DefaultAge(23));
```

Encoder

```
1  package java.net;
2
3  class URLEncoder {
4    static String encode(String s, String enc) {
5      // Encode the string "s" using
6      // the "enc" encoding and return
7      // the encoded string
8    }
9  }
10
11 String e = URLEncoder.encode("@foo");
12 e.equals("%40foo");
```

```
1  class Encoded implements String {
2    private final String origin;
3    private final String encoding;
4    Encoded(String s, String enc) {
5      this.origin = s;
6      this.enc = encoding;
7    }
8    @Override String value() {
9      // Encode the string "origin" using
10     // the "encoding" and return
11     // the encoded string
12   }
13 }
14
15 String e = new Encoded("@foo");
16 e.value().equals("%40foo");
```

The right snippet won't work in Java, since `String` is a final class, not an interface, unfortunately.

Chapter #2:
-Client Suffix

AWS Java Client

```
 1  class AmazonS3Client {
 2    createBucket(String name);
 3    deleteBucket(String name);
 4    doesBucketExist(String name);
 5    getBucketAcl(String name)
 6    getBucketPolicy(String name);
 7    listBuckets();
 8    // 160+ more methods here
 9  }
10  client = new AmazonS3Client("us-1");
11  client.createBucket("foo");
12  client.putObject("foo", "a.txt");
13  client.writeObject("foo", "a.txt", "data");
```

```
 1  region = new S3Region("us-1");
 2  bucket = region.createBucket("foo");
 3  object = bucket.putObject("a.txt");
 4  object.write("data");
```

The left snippet is: 1) procedural, 2) hard to test, 3) resembles a utility class, and 4) is hard to extend. The right one is object-oriented.

Chapter #3:
What About Performance?

Sticky Parser

```
1  class StringAsInt implements Number {
2    private final String txt;
3    StringAsInt(String t) { this.txt = t; }
4    @Override int intValue() {
5      // Parse String into Integer
6      // and return the value
7    }
8  }
9
10 Number n = new StringAsInt("42");
11 int x = n.intValue();
```

```
1  class StickyInt implements Number {
2    private final Number origin;
3    private int cache = 0;
4    private bool cached = false;
5    StickyInt(Number n) { origin = n; }
6    @Override int intValue() {
7      if (!cached) {
8        cache = origin.intValue();
9      }
10     return cache;
11   }
12 }
```

Is it thread-safe though?

Thread-safe Sticky Parser

```
1  class StickyInt implements Number {
2    private final Number origin;
3    private int cache = 0;
4    private bool cached = false;
5    StickyInt(Number n) { origin = n; }
6    @Override int intValue() {
7      if (!cached) {
8        cache = origin.intValue();
9      }
10     return cache;
11   }
12 }
```

```
1  class StickyInt implements Number {
2    private final Number origin;
3    private final AtomicReference<Integer> cache =
4      new AtomicReference<Integer>(null);
5    StickyInt(Number n) { origin = n; }
6    @Override int intValue() {
7      return cache.updateAndGet(
8        x -> {
9          if (x == null) {
10           return origin.intValue();
11         }
12         return x;
13       }
14     );
15   }
16 }
```
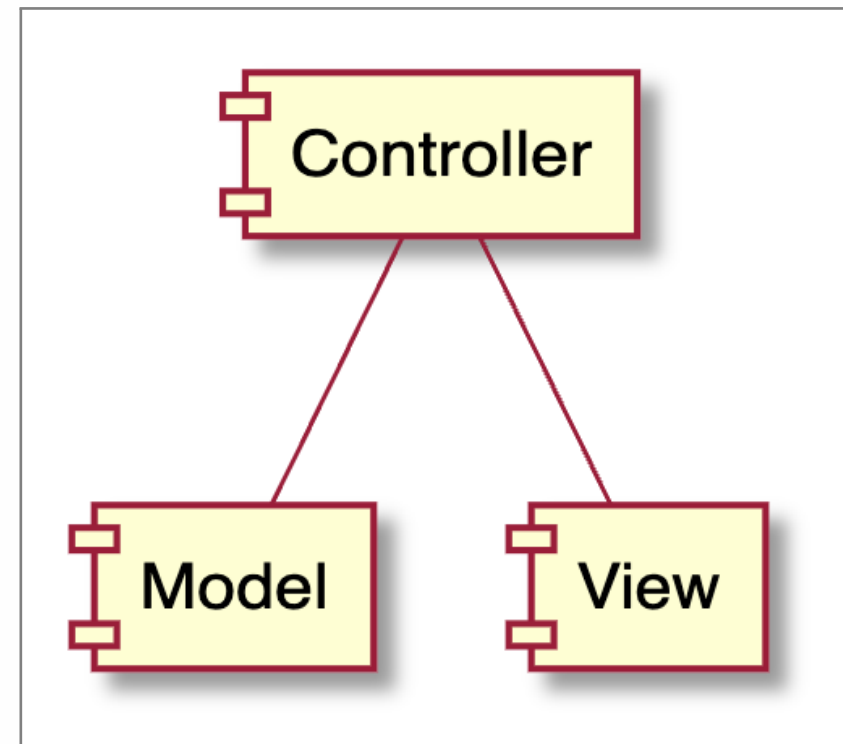
The left snippet is not thread-safety, while the right one is.

Chapter #4:

Model-View-Controller (MVC)

The Controller

```
1  class Controller {
2    @GET
3    @Path("/b{id}")
4    String index(int id) {
5      Book book = em.findById(id);
6      View v = new HtmlView("book.html");
7      v.set("title", book.getTitle());
8      return v.renderHtml();
9    }
10 }
```

Book as HTML

```
1  class Controller {
2    @GET
3    @Path("/b{id}")
4    String index(int id) {
5      Book book = em.findById(id);
6      View v = new HtmlView("book.html");
7      v.set("title", book.getTitle());
8      return v.renderHtml();
9    }
10 }
```

```
1  interface Book
2    String title();
3  class PgBook implements Book
4    String title() // loads from PostgreSQL
5  interface Page
6    String html();
7  class HtmlBook implements Book, Page
8    String html() // renders in HTML
9    String title() // returns origin.title()
10 class PageOnPath implement Page
11   private final String path;
12   private final Page origin;
13   String html() // renders if path matches
```

Check yegor256/jpages and yegor256/takes.

Chapter #5:
Read and Watch

Don't Create Objects That End With -ER by me

Yet Another Evil Suffix For Object Names: Client by me

MVC vs. OOP by me