-ER

...

Yegor Bugayenko

Lecture #5 out of 8
90 minutes

Examples and Alternatives

-Client Suffix

What About Performance?

Read and Watch

“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

```
class Parser {
  static int parseInt(String t) {
    // Parse String into Integer
  }
  static float parseFloat(String t) {
    // Parse String into Float
  }
  // And many more methods...
}

int x = Parser.parseInt("42");
```

```
class StringAsInt implements Number {
  private final String txt;
  StringAsInt(String t) { this.txt = t; }
  @Override int intValue() {
    // Parse String into Integer
    // and return the value
  }
}

Number n = new StringAsInt("42");
int x = n.intValue();
```

Reader

```
class Reader {
  static char readChar(InputStream i) {
    // Read the next char from the
    // stream and return it, or NULL
    // if the stream is at the EOF
  }
}

InputStream i = new FileInputStream(..);
char c = Reader.readChar(i);
```

```
class Chars
  private final InputStream is;
  Chars(InputStream i)
    this.is = i;
  char next()
    // Read the next char from the
    // stream and throw exception
    // if !exists()
  bool exists()
    // Return TRUE if not EOF

InputStream i = new FileInputStream(..);
Chars chars = new Chars(i);
char c = chars.next();
```

Controller

```
class SimpleController {
  @GET
  @Path("/index")
  HttpResponse index(HttpRequest e) {
    // Build an index page and return
  }
  @POST
  @Path("/update")
  HttpResponse update(HttpRequest e) {
    // Save new user information
    // and return HTTP 303
  }
}
```

```
class IndexPage implements HttpPage
  HttpResponse process(HttpRequest e) {
    // Build an index page and return
  }
class UpdatePage implements HttpPage
  HttpResponse process(HttpRequest e) {
    // Save new user information
    // and return HTTP 303
  }

new AllPages(
  new IndexPage(),
  new UpdatePage()
);
```

Validator

```
class Validator {
  bool isValid(int age) {
    return age >= 18;
  }
}
int a = 23;
Validator v = new Validator();
if (!v.isValid(a)) {
  throw new Exception(
    "Age is not valid"
  );
}
```

```
interface Age
  int value();
class DefaultAge implements Age
  private final int a;
  DefaultAge(int a)
    this.a = a;
  @Override int value()
    return this.a;
class ValidAge implements Age {
  private final Age origin;
  ValidAge(Age age)
    this.origin = age;
  @Override int value()
    int v = this.origin.value();
    if (v < 18)
      throw new Exception("Age is not valid");
    return v;

Age a = new ValidAge(new DefaultAge(23));
```

Encoder

```
package java.net;

class URLEncoder {
  static String encode(String s, String enc) {
    // Encode the string "s" using
    // the "enc" encoding and return
    // the encoded string
  }
}

String e = URLEncoder.encode("@foo");
e.equals("%40foo");
```

```
class Encoded implements String {
  private final String origin;
  private final String encoding;
  Encoded(String s, String enc) {
    this.origin = s;
    this.enc = encoding;
  }
  @Override String value() {
    // Encode the string "origin" using
    // the "encoding" and return
    // the encoded string
  }
}

String e = new Encoded("@foo");
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

```
class AmazonS3Client {
  createBucket(String name);
  deleteBucket(String name);
  doesBucketExist(String name);
  getBucketAcl(String name)
  getBucketPolicy(String name);
  listBuckets();
  // 160+ more methods here
}
client = new AmazonS3Client("us-1");
client.createBucket("foo");
client.putObject("foo", "a.txt");
client.writeObject("foo", "a.txt", "data");
```

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

```
class StringAsInt implements Number {
  private final String txt;
  StringAsInt(String t) { this.txt = t; }
  @Override int intValue() {
    // Parse String into Integer
    // and return the value
  }
}

Number n = new StringAsInt("42");
int x = n.intValue();
```

```
class StickyInt implements Number {
  private final Number origin;
  private int cache = 0;
  private bool cached = false;
  StickyInt(Number n) { origin = n; }
  @Override int intValue() {
    if (!cached) {
      cache = origin.intValue();
    }
    return cache;
  }
}
```

Is it thread-safe though?

Thread-safe Sticky Parser

```
class StickyInt implements Number {
  private final Number origin;
  private int cache = 0;
  private bool cached = false;
  StickyInt(Number n) { origin = n; }
  @Override int intValue() {
    if (!cached) {
      cache = origin.intValue();
    }
    return cache;
  }
}
```

```
class StickyInt implements Number {
  private final Number origin;
  private final AtomicReference<Integer> cache =
    new AtomicReference<Integer>(null);
  StickyInt(Number n) { origin = n; }
  @Override int intValue() {
    return cache.updateAndGet(
      x -> {
        if (x == null) {
          return origin.intValue();
        }
        return x;
      }
    );
  }
}
```

The left snippet is not thread-safety, while the right one is.

Chapter #4:

Read and Watch

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

Yet Another Evil Suffix For Object Names: Client by me