Reflection

Classpath, Casting, Annotations

Yegor Bugayenko

Lecture #7 out of 8
90 minutes

- Type Casting
- Classpath Scanning
- Annotations
- Read and Watch

Chapter #1:

Type Casting

Iterable → Collection

```
1  import java.lang.Collection;
2  import java.lang.Iterable;
3
4  int sizeOf(Iterable items) {
5    int size = 0;
6    if (items instanceof Collection) {
7      size = ((Collection) items).size();
8    } else {
9      for (Object item : items) {
10       ++size;
11     }
12   }
13   return size;
14 }
```

```
1  int sizeOf(Iterable items) {
2    int size = 0;
3    for (Object item : items) {
4      ++size;
5    }
6    return size;
7  }
8  int sizeOf(Collection items) {
9    return items.size();
10 }
```

Chapter #2:

Classpath Scanning

Finding Java classes

```
interface Foo {}

class Bar implements Foo {}

Reflections rts =
  new Reflections("");
Set<Class<?>> types = rts.get(
  SubTypes.of(Foo.class).asClass()
);
```

```
public @interface Foo {}

@Foo
class Bar {}

Reflections rts =
  new Reflections("");
Set<Class<?>> types = rts.get(
  SubTypes.of(
    TypesAnnotated.with(Foo.class)
  ).asClass()
);
```

The library is called Reflections.

Chapter #3:

Annotations

I lieu of static methods

```
interface Pub
  String isbn();

class Book implements Pub
  @Override public String isbn()
    /* ... */
  public static String category()
    return "book";

class Journal implements Pub
  @Override public String isbn()
    /* ... */
  public static String category()
    return "journal";
```

```
interface Pub
  String isbn();

@Target(ElementType.CLASS)
@Retention(RetentionPolicy.SOURCE)
public @interface Category
  String value();

@Category("book")
class Book implements Pub
  @Override public String isbn()
    /* ... */

@Category("journal")
class Journal implements Pub
  @Override public String isbn()
    /* ... */
```

Locating methods

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.SOURCE)
public @interface Path
  String url;

class BookController
  @Path("/book-title")
  String title()
    // Build HTML page and return it
```

```
String dispatch(String url) {
  c = new BooksController();
  b = BooksController.class;
  for (Method m : b.getDeclaredMethods()) {
    if (m.isAnnotationPresent(Path.class)) {
      Annotation a = m.getAnnotation(Path.class);
      if (a.url().equals(url)) {
        return m.invoke(c);
      }
    }
  }
  return "404 Page not found";
}
```

Dependency Injection Container

```
interface Shipment
  int cost();

class Cart
  @Inject private Shipment shmt;
  private Book book;
  void setBook(Book b)
    this.book = b;
  int cost()
    return this.book.price() + this.shmt.cost();

container = new Container();
c = container.make(Cart.class);
c.setBook(new Book("1984"));
x = c.cost();
```

```
class Container {
  private HashMap<Class, Object> cache =
    new ConcurrentHashMap<>();
  T make(Class<T> type) {
    // 1. Find @Inject-annotated "shmt" field;
    // 2. Make an instance of "Shipment";
    // 3. Store it in the "cache";
    // 4. Make an instance of "Cart";
    // 5. Store "cart" in the "cache";
    // 6. Assign "shipment" to "cart.shmt";
    // 7. Return "cart".
  }
}
```

How do you think, at the step no.2, what class will be instantiated?

Dependency Injection *without* a Container

```
interface Shipment
  int cost();

class Cart
  @Inject private Shipment shmt;
  private Book book;
  void setBook(Book b)
    this.book = b;
  int cost()
    return this.book.price() + this.shmt.cost();

container = new Container();
c = container.make(Cart.class);
c.setBook(new Book("1984"));
x = c.cost();
```

```
interface Shipment
  int cost();

class Cart
  private final Shipment shmt;
  private final Book book;
  Cart(Shipment s, Book b)
    this.shmt = s;
    this.book = b;
  int cost()
    return this.book.price() + this.shmt.cost();

c = new Cart(new MyShipment(), new Book("1984"));
x = c.cost();
```

Chapter #4:

Read and Watch

Dependency Injection Containers are Code Polluters by me (2014)

Class Casting Is a Discriminating Anti-Pattern by me (2015)

Java Annotations Are a Big Mistake by me (2016)

Reflection Means Hidden Coupling by me (2022)

Java Annotations Are a Bad Idea, at JDK.io conference (2017)