Ingegneria del Software

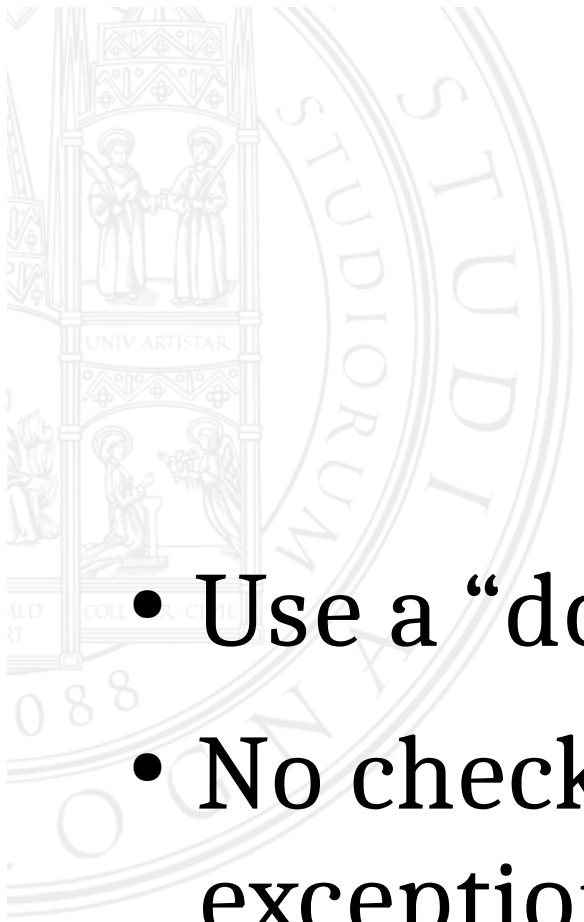
Corso di Laurea in Informatica per il Management

Modern patterns

Davide Rossi
Dipartimento di Informatica
Università di Bologna

Null Object

- The problem
  - null a value on a non-value?
  - When we pass or return null where a reference in required/expected we mean “default value” or “no object”?
  - If null can be passed/returned we have to check for that to avoid errors/exceptions
- Tony Hoare introduced null references in ALGOL W back in 1965. It later called it “The Billion Dollar Mistake”.



Null Object

- Use a “do nothing” object instead of null
- No check-for-null; no null pointer exception
- No standard solution

Null Object in Java

```
public interface Cat {
   public static final Cat NONE =
      new Cat() {
         //"do-noting" methods
         Cat getParent() {
            return NONE;
         }
      };
}
…
Cat grandParent =
      cat.getParent().getParent();
```

The Hollywood principle

- “Don't call us, we'll call you”
- Who controls who?
- Library or framework?
- IoC: inversion of control

Dependency Injection

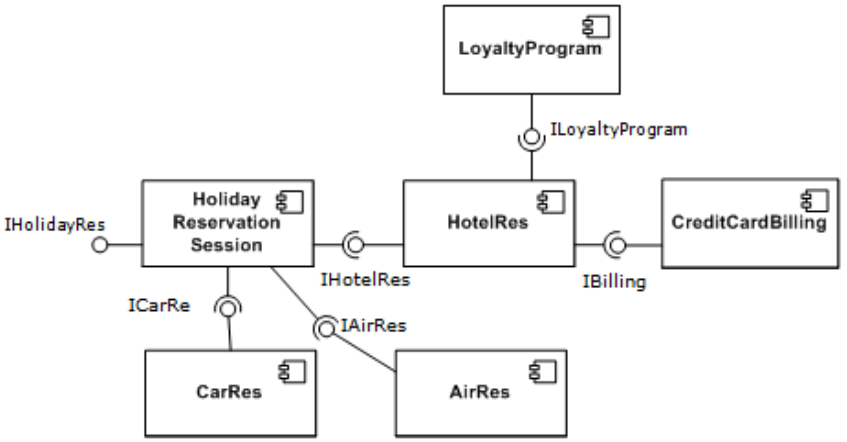
- A design pattern which is an application of IoC
- Dependency injection creates a graph of dependencies by inverting the control to find associated objects, *pushing dependencies from the core to the edges* - up to a *composition root*
- Break the dependency from new/factories ("the new new")
- Greatly simplifies **testing**, improves **modularity**.

Types of injection

- Constructor injection
  - Dependencies are provided via constructor parameters
- Setter injection (also: field injection)
  - Dependencies are provided by calling specific setters
- Interface injection
  - Injection methods are declared in interfaces, a class implements an injection interface for each dependency to inject
- Method injection
  - Used when a dependency can be variously solved on a per-operation basis (perform this operation using this service)

Component based architectures

- In component-based software engineering (CBSE), software systems are built by gluing together software components on the basis of *provided* and *required* interfaces



DI and CBSE

- By exposing the dependencies to be injected in its interface an object exposes both what it *provides* and what it *requires*, easing a component based approach to software design.

“Clean” DI

- Use constructor injection for dependencies always needed that do not change for the lifetime of the instance
- Use method injection for dependencies that are needed only during the invocation of that method

“Clean” DI

- When binding has to be solved at run-time pass factories to constructors or methods.
  - Try to isolate choice points in strategy-like structures
  - If the language allows, let the dependency emerge from the signature of the factory.
    For example, in Java, make factories implement a Factory<Dependency> interface.

DI in Java

- Pure DI, no framework, the composition root is *close to the entry point(s) of the application*
- DI frameworks: Guice, Dagger, Spring, CDI, …
- Annotations are used to mark dependencies that have to be injected

Example

```
public void postButtonClicked() {
    String text = textField.getText();
    if (text.length() > 140) {
      Shortener shortener = new TinyUrlShortener();
      text = shortener.shorten(text);
    }
    if (text.length() <= 140) {
      Tweeter tweeter = new SmsTweeter();
      tweeter.send(text);
      textField.clear();
    }
}
```

From Google I/O 09

Example

```
public class TweetClient {
  private final Shortener shortener;
  private final Tweeter tweeter;
  public TweetClient(Shortener shortener,
                     Tweeter tweeter) {
    this.shortener = shortener;
    this.tweeter = tweeter;
  }
  public void postButtonClicked() {
    ...
    if (text.length() <= 140) {
      tweeter.send(text);
      textField.clear();
    }
```

Example with Guice

```
import com.google.inject.Inject;

public class TweetClient {
  private final Shortener shortener;
  private final Tweeter tweeter;

  @Inject
  public TweetClient(Shortener shortener,
                     Tweeter tweeter) {
    this.shortener = shortener;
    this.tweeter = tweeter;
  }
```

Example with Guice

```
import com.google.inject.AbstractModule;

public class TweetModule extends AbstractModule {
  protected void configure() {
    bind(Tweeter.class).to(SmsTweeter.class);
    bind(Shortener.class).to(TinyUrlShortener.class);
  }
}
```

Example with Guice

```
public static void main(String[] args) {
  Injector injector
      = Guice.createInjector(new TweetModule());
  TweetClient tweetClient
      = injector.getInstance(TweetClient.class);
  tweetClient.show();
}
```