# Spring

## Inversion of Control

# RECAP

```
public class CountryController {
    private Service service = new HolidayService();

    public                  getCountries() {
        return
    }

}
```

# Dependency Inversion Principle

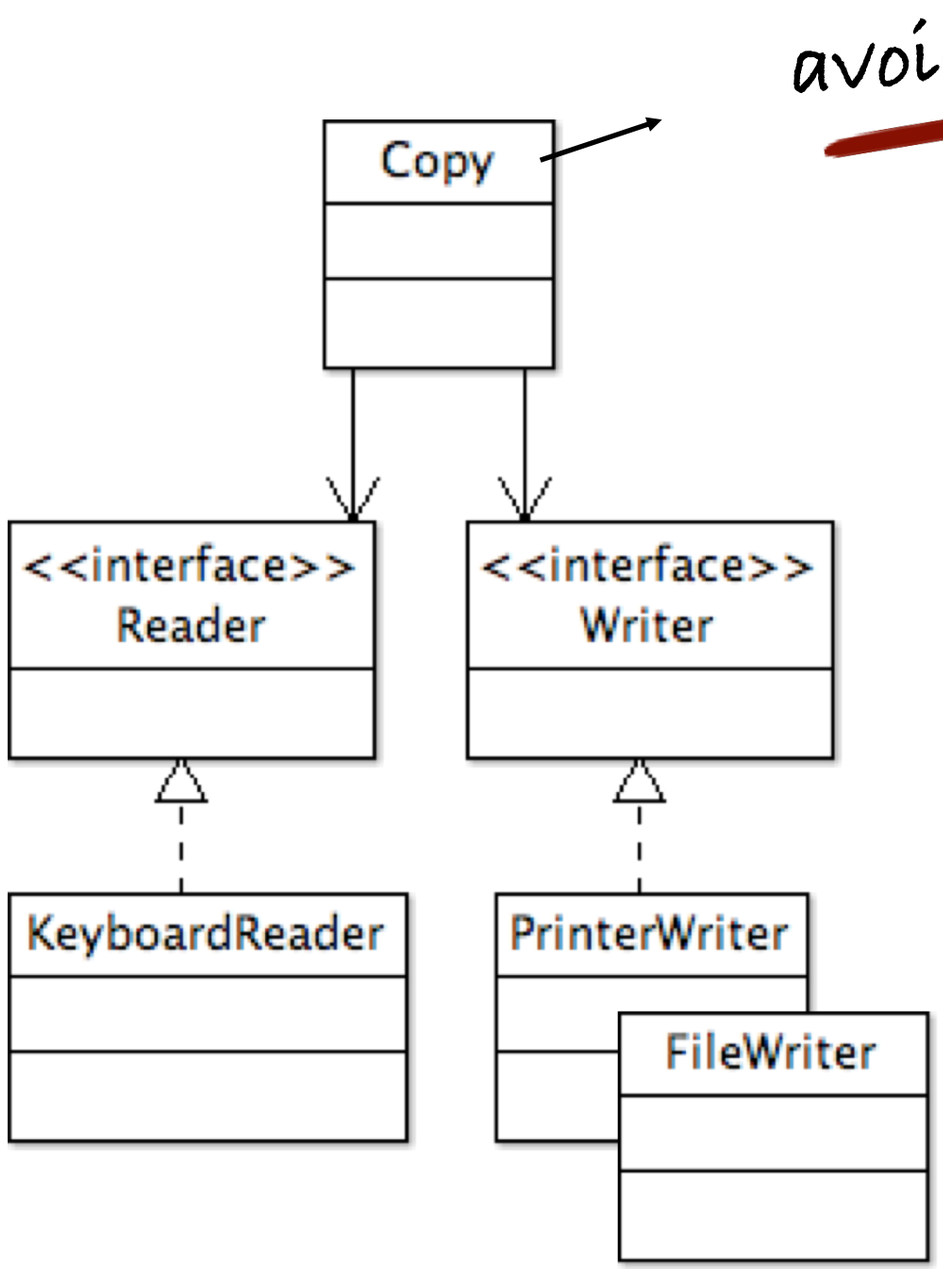
***"High-level modules should not depend on low-level modules. Both should depend on abstractions."***

# BETTER

***High-level class:***

***Abstract layer in between:***

***Low-level classes:***



# DEPENDENCIES

Disadvantage?

service is a **dependency** of CountryController
CountryController **depends** on service

```
@Controller
@RequestMapping(value = "/country")
public class CountryController {
    private Service service = new HolidayService();

    @RequestMapping(method = RequestMethod.GET)
    public ModelAndView getCountries() {
        return new ModelAndView("countries", "countries", service.getCountries());
    }

}
```

# Inversion of Control

## By developer

```
@Controller
@RequestMapping(value = "/country")
public class CountryController {
    private Service service;

    public CountryController(Service service) {
        this.service = service;
    }

    @RequestMapping(method = RequestMethod.GET)
    public ModelAndView getCountries() {
        return new ModelAndView("countries", "countries", service.getCountries());
    }
}
```

No initialisation here

Dependency is created elsewhere

Control is in other class

```
public class ServiceFactory {
    public Service createService(String type){
        Service service;

        if(type.equals("holiday")) {
            service = new HolidayService();
        } else {
```

Dependency is created here and passed as parameter to constructor

Done by developer

# Inversion of Control

## By Spring

```
@Controller
@RequestMapping(value = "/country")
public class CountryController {
    private Service service;

    public CountryController(@Autowired Service service) {
        this.service = service;
    }

    @RequestMapping(method = RequestMethod.GET)
    public ModelAndView getCountries() {
        return new ModelAndView("countries", "countries", service.getCountries());
    }
}
```

Dependency is created and **injected** by Spring

How does the container know which type?

# ApplicationConfig

```
package be.ucll.tourism.web.config;

import be.ucll.tourism.service.HolidayService;
import be.ucll.tourism.service.TourismService;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class ApplicationConfig {

    @Bean
    public TourismService service() {
        return new HolidayService("Memory");
    }
}
```

name of the object

type of the object

argument for the constructor

# SPRING CONFIG FILES*

- DispatcherServletConfig
  - configuration of your **servlet**
  - beans can read beans defined in applicationContext.xml

X

V

- ApplicationConfig
  - general configuration for the **application**
  - beans can NOT read beans defined in dispatcher-servlet.xml

*can be replaced by annotations

# SPRING BEAN

*The objects that form the backbone of your application and that are managed by the Spring IoC container are called beans. A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container. These beans are created with the configuration metadata that you supply to the container, …*

- *countryController* is a bean
  *org.ucll.demo.controller.CountryController.java*
- *service* is a bean
  *org.ucll.demo.domain.HolidayService.java*
- *indexController* is a bean
  *org.springframework.web.servlet.mvc.ParameterizableViewController*
- *viewResolver* is a bean
  *org.springframework.web.servlet.view.InternalResourceViewResolver*
- …

REMINDER

# JavaBean

```
public class Country {
    private String name;

    public Country() {
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

public no-arg constructor

getter and setter
define a property

# SPRING

## AND OTHER FRAMEWORKS

also called **dependency injection** (DI)

- Inversion of control (IoC)
- Objects ***define*** their dependencies
  - Spring: `@Autowired`
- Container:
  - create and manages beans
  - ***injects*** dependencies when creating a bean

# Inversion of Control Dependency Injection