Internet Applications Design and Implementation 2017/2018 (Lab 7: Storage in Spring – H2 in memory)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Leitão (jc.leitao@fct.unl.pt)
João Costa Seco (joao.seco@fct.unl.pt)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Lab8 Goal:

- Introduction to Storage in Spring.
- Example and Running the Example.
- Project development (materialize your application state on a storage layer).

Storage
in Java Spring

- Similar to other aspects in Spring (such as defining which controllers export REST services for clients) storage is highly abstracted by the Framework.
- More importantly:
  - The framework promotes an easy way for switching which data storage solution your application uses.
  - This is achieved through three complementary strategies:
  1. Abstraction of the storage layer.
  2. Isolation of the storage layer behavior.
  3. Dynamic biding of components.

1. Abstraction of the storage layer

- Data objects (that can be stored and accessed) have properties that define how they are stored.

- These properties are established through annotations that are agnostic to the concrete storage that you are using:
  - in memory (H2)
  - SQL based storage (MySQL)
  - Distributed Key-Value Store (Cassandra)
  - …

2. Isolation of the storage layer behavior

- All interactions with the storage layer are performed through a generic interface.

- The interface itself only exposes the mechanisms used to manipulate the data storage layer (e.g, get a data object, create a data object, execute a query to obtain multiple data objects).
- The interface however does not have any concrete reference to a storage layer.

3. Dynamic biding of components

- Defining the connection between the interface used by the application and a concrete storage system is executed at runtime by the Spring framework.

- An annotation explicitly requests this dynamic biding to be performed.

- The biding itself is doe based on the execution environment conditions.

Storage in Spring
by example…

- We will now see how this works in practice:

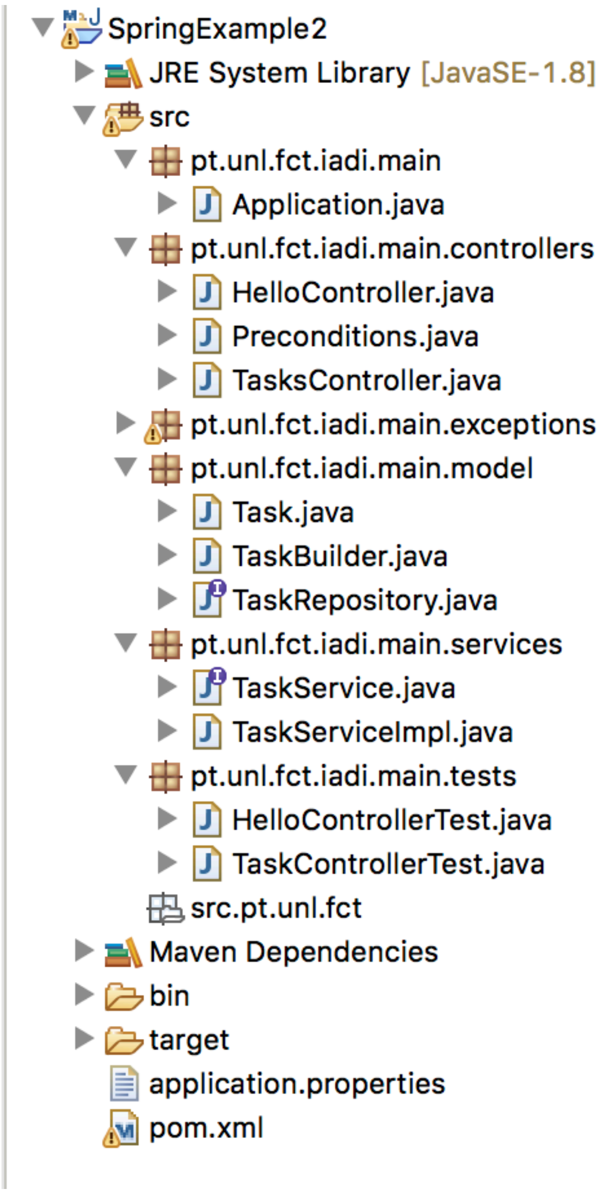
- Enrich last week example to deal with a storage layer (e.g, getting read of that erroneous hash table in the Tasks Controller).
- We will use today a in memory data storage commonly used for development and testing (H2).
- H2 will not offer persistence (every time you restart you application the database is reset)
- In the final delivery of the project you are expected to deliver everything working on top of MySQL (SQL interface with persistence) this will be covered in the Lectures.

Storage in Spring
by example…

- Some as last week:
- Go to the public repository of CIAI:
- https://bitbucket.org/costaseco/ciai-1718-public

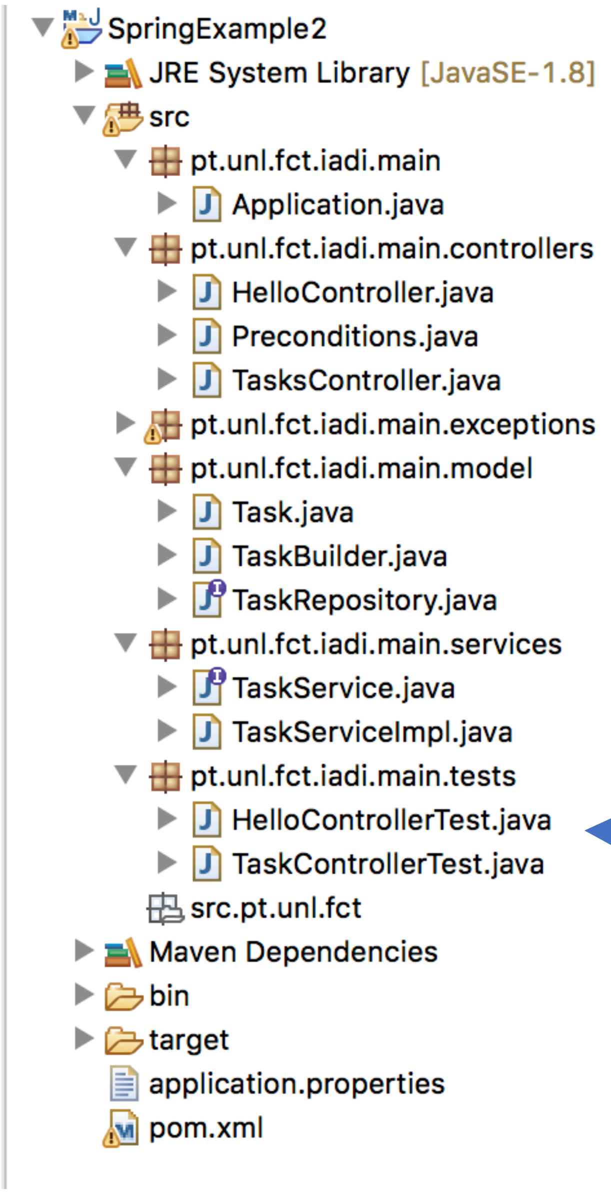
- Update the repository.
- You should have a “Project” named SpringExample2
- Import it into your favorite Java IDE
- Convert it to a Maven project.

Spring by example...



```
SpringExample2
  JRE System Library [JavaSE-1.8]
  src
    pt.unl.fct.iadi.main
      Application.java
    pt.unl.fct.iadi.main.controllers
      HelloController.java
      Preconditions.java
      TasksController.java
    pt.unl.fct.iadi.main.exceptions
    pt.unl.fct.iadi.main.model
      Task.java
      TaskBuilder.java
      TaskRepository.java
    pt.unl.fct.iadi.main.services
      TaskService.java
      TaskServiceImpl.java
    pt.unl.fct.iadi.main.tests
      HelloControllerTest.java
      TaskControllerTest.java
    src.pt.unl.fct
  Maven Dependencies
  bin
  target
  application.properties
  pom.xml
```

Spring by example…

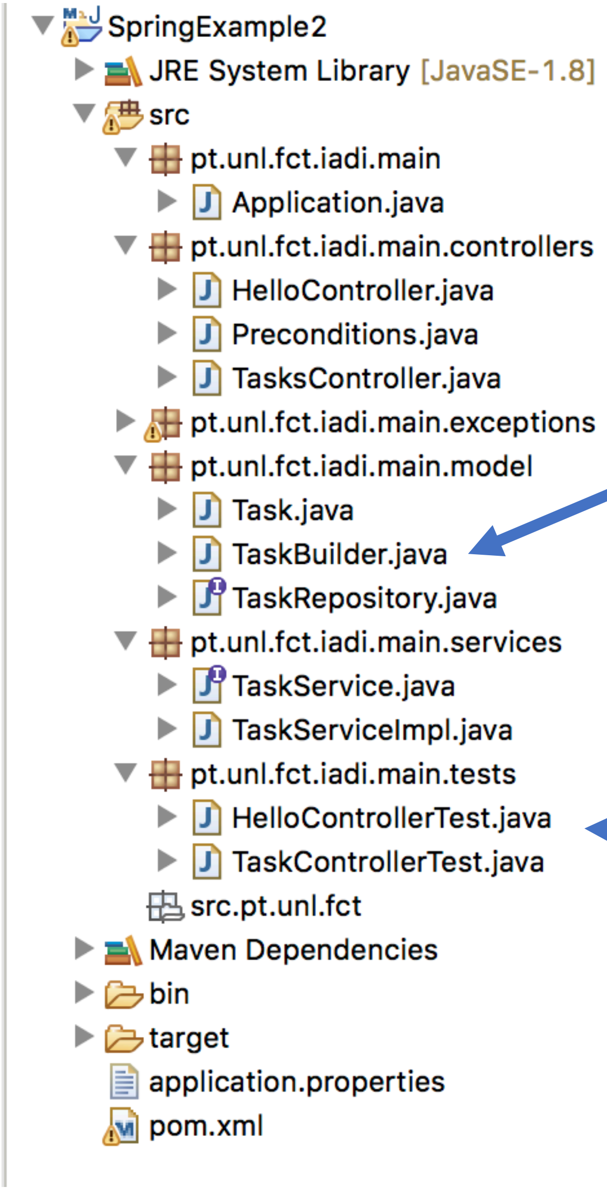


```
▼ SpringExample2
  ▶ JRE System Library [JavaSE-1.8]
  ▼ src
    ▼ pt.unl.fct.iadi.main
      ▶ Application.java
    ▼ pt.unl.fct.iadi.main.controllers
      ▶ HelloController.java
      ▶ Preconditions.java
      ▶ TasksController.java
    ▶ pt.unl.fct.iadi.main.exceptions
    ▼ pt.unl.fct.iadi.main.model
      ▶ Task.java
      ▶ TaskBuilder.java
      ▶ TaskRepository.java
    ▼ pt.unl.fct.iadi.main.services
      ▶ TaskService.java
      ▶ TaskServiceImpl.java
    ▼ pt.unl.fct.iadi.main.tests
      ▶ HelloControllerTest.java
      ▶ TaskControllerTest.java
    src.pt.unl.fct
  ▶ Maven Dependencies
  ▶ bin
  ▶ target
    application.properties
    pom.xml
```

These are test tools… we are not going to discuss them today yet! (João Costa Seco will address this in the lecture)

Spring by example…

spring by Pivotal

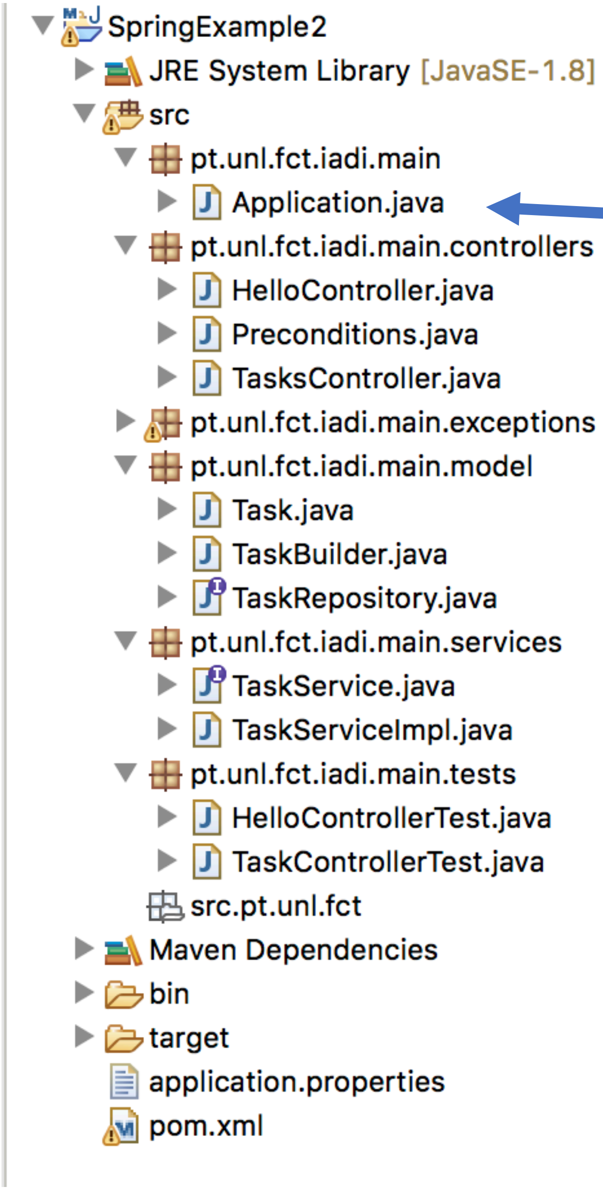


```
▼ SpringExample2
  ▶ JRE System Library [JavaSE-1.8]
  ▼ src
    ▼ pt.unl.fct.iadi.main
      ▶ Application.java
    ▼ pt.unl.fct.iadi.main.controllers
      ▶ HelloController.java
      ▶ Preconditions.java
      ▶ TasksController.java
    ▶ pt.unl.fct.iadi.main.exceptions
    ▼ pt.unl.fct.iadi.main.model
      ▶ Task.java
      ▶ TaskBuilder.java
      ▶ TaskRepository.java
    ▼ pt.unl.fct.iadi.main.services
      ▶ TaskService.java
      ▶ TaskServiceImpl.java
    ▼ pt.unl.fct.iadi.main.tests
      ▶ HelloControllerTest.java
      ▶ TaskControllerTest.java
    src.pt.unl.fct
  ▶ Maven Dependencies
  ▶ bin
  ▶ target
    application.properties
    pom.xml
```

The TaskBuilder class wraps a task with methods to manipulate such task. This uses the Builder patters (also for tests)

These are test tools… we are not going to discuss them today yet! (João Costa Seco will address this in the lecture)

Spring by example...

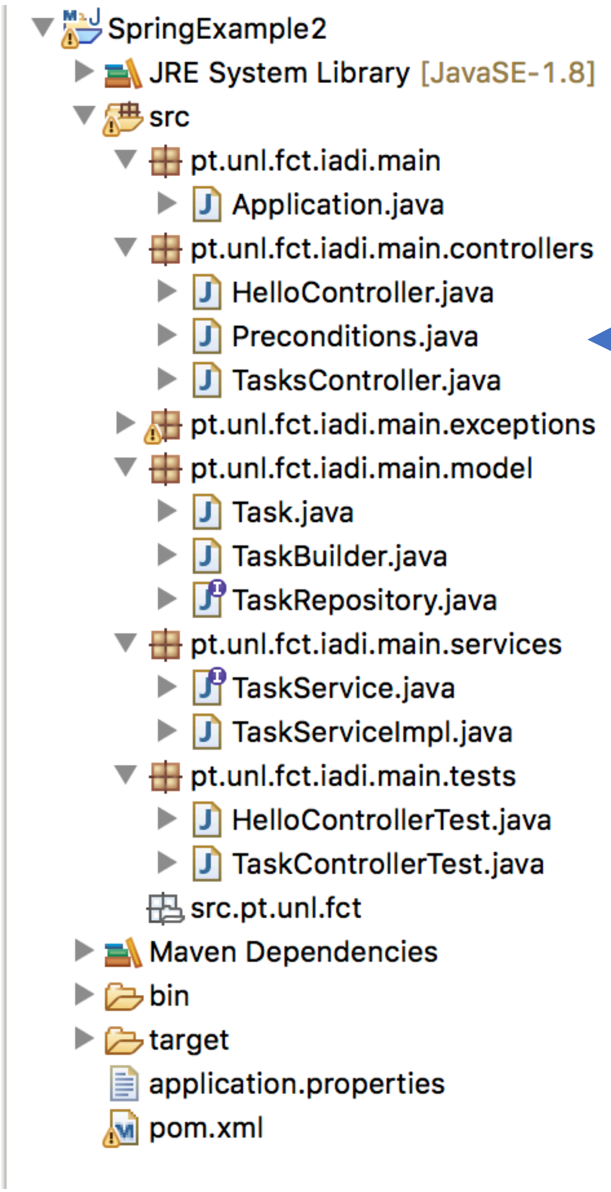

SpringExample2
JRE System Library [JavaSE-1.8]
src
pt.unl.fct.iadi.main
Application.java
pt.unl.fct.iadi.main.controllers
HelloController.java
Preconditions.java
TasksController.java
pt.unl.fct.iadi.main.exceptions
pt.unl.fct.iadi.main.model
Task.java
TaskBuilder.java
TaskRepository.java
pt.unl.fct.iadi.main.services
TaskService.java
TaskServiceImpl.java
pt.unl.fct.iadi.main.tests
HelloControllerTest.java
TaskControllerTest.java
src.pt.unl.fct
Maven Dependencies
bin
target
application.properties
pom.xml

Some of the suff we saw last week do not require modifications at all.

- Application (although we now have something there for testing)

Spring by example…

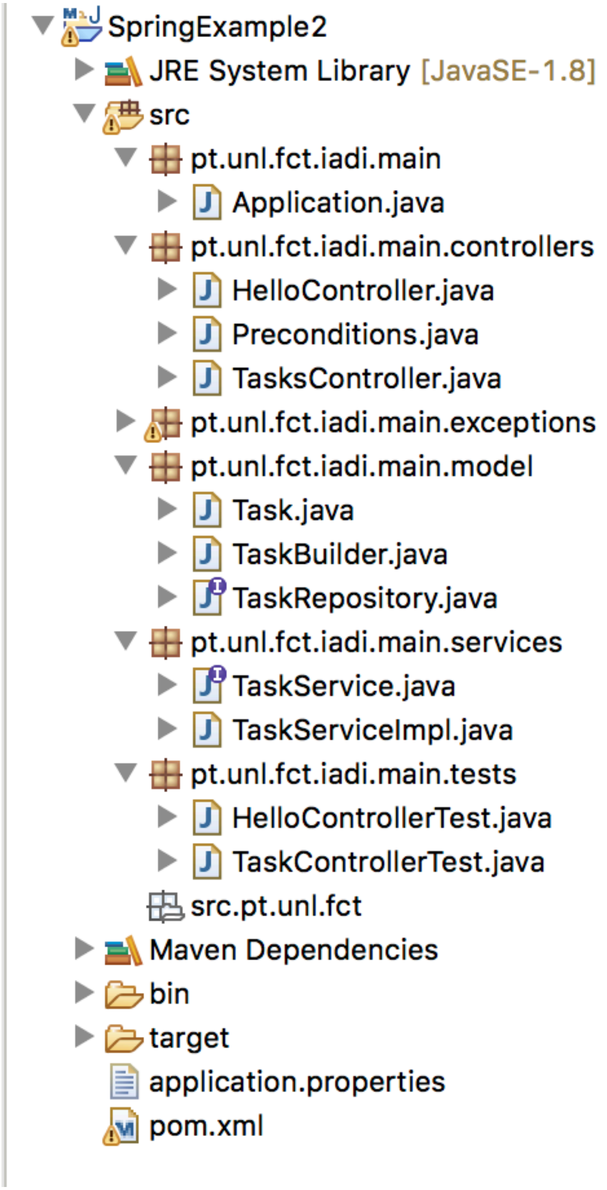
spring
by Pivotal



```
▼ SpringExample2
  ▶ JRE System Library [JavaSE-1.8]
  ▼ src
    ▼ pt.unl.fct.iadi.main
      ▶ Application.java
    ▼ pt.unl.fct.iadi.main.controllers
      ▶ HelloController.java
      ▶ Preconditions.java
      ▶ TasksController.java
    ▶ pt.unl.fct.iadi.main.exceptions
    ▼ pt.unl.fct.iadi.main.model
      ▶ Task.java
      ▶ TaskBuilder.java
      ▶ TaskRepository.java
    ▼ pt.unl.fct.iadi.main.services
      ▶ TaskService.java
      ▶ TaskServiceImpl.java
    ▼ pt.unl.fct.iadi.main.tests
      ▶ HelloControllerTest.java
      ▶ TaskControllerTest.java
    src.pt.unl.fct
  ▶ Maven Dependencies
  ▶ bin
  ▶ target
    application.properties
    pom.xml
```

These are Controllers interface was not modified, however we do have to make small adjustments to ensure that we no longer use the local HashTable and instead use a (generic) storage interface.

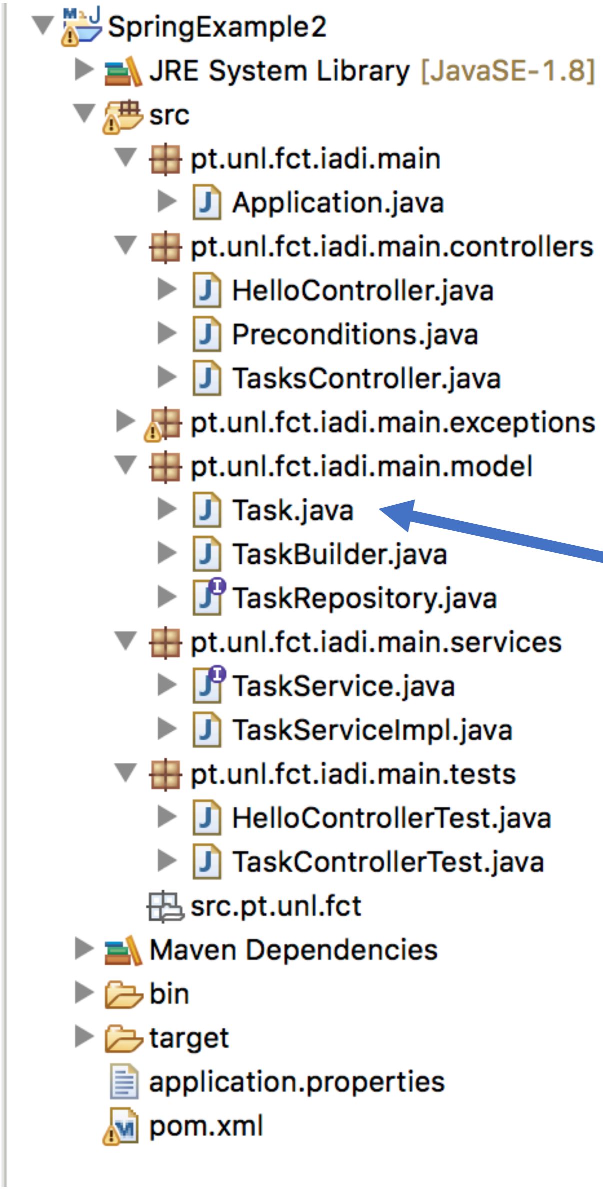
Spring by example...



```
SpringExample2
  JRE System Library [JavaSE-1.8]
  src
    pt.unl.fct.iadi.main
      Application.java
    pt.unl.fct.iadi.main.controllers
      HelloController.java
      Preconditions.java
      TasksController.java
    pt.unl.fct.iadi.main.exceptions
    pt.unl.fct.iadi.main.model
      Task.java
      TaskBuilder.java
      TaskRepository.java
    pt.unl.fct.iadi.main.services
      TaskService.java
      TaskServiceImpl.java
    pt.unl.fct.iadi.main.tests
      HelloControllerTest.java
      TaskControllerTest.java
    src.pt.unl.fct
  Maven Dependencies
  bin
  target
  application.properties
  pom.xml
```

No changes in the Exceptions...

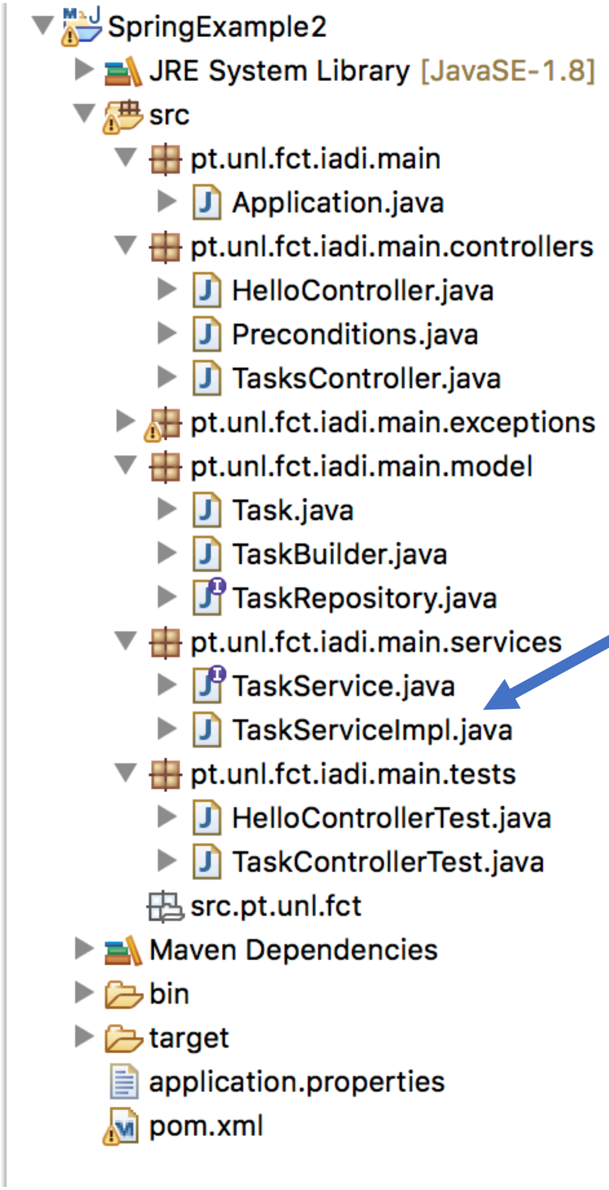
Spring by example…



```
SpringExample2
  JRE System Library [JavaSE-1.8]
  src
    pt.unl.fct.iadi.main
      Application.java
    pt.unl.fct.iadi.main.controllers
      HelloController.java
      Preconditions.java
      TasksController.java
    pt.unl.fct.iadi.main.exceptions
    pt.unl.fct.iadi.main.model
      Task.java
      TaskBuilder.java
      TaskRepository.java
    pt.unl.fct.iadi.main.services
      TaskService.java
      TaskServiceImpl.java
    pt.unl.fct.iadi.main.tests
      HelloControllerTest.java
      TaskControllerTest.java
    src.pt.unl.fct
  Maven Dependencies
  bin
  target
  application.properties
  pom.xml
```

The Task class represents an entity of our data model that can be stored and accessed by the application. We do have to add some details here.

Spring by example…



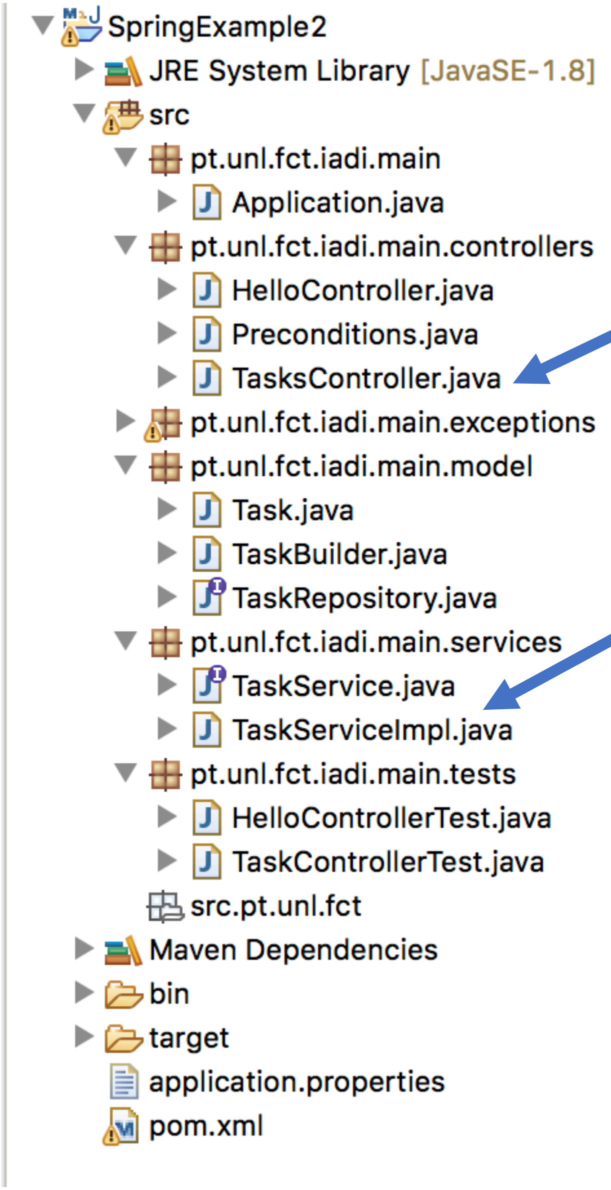
```
▼ SpringExample2
  ▶ JRE System Library [JavaSE-1.8]
  ▼ src
    ▼ pt.unl.fct.iadi.main
      ▶ Application.java
    ▼ pt.unl.fct.iadi.main.controllers
      ▶ HelloController.java
      ▶ Preconditions.java
      ▶ TasksController.java
    ▶ pt.unl.fct.iadi.main.exceptions
    ▼ pt.unl.fct.iadi.main.model
      ▶ Task.java
      ▶ TaskBuilder.java
      ▶ TaskRepository.java
    ▼ pt.unl.fct.iadi.main.services
      ▶ TaskService.java
      ▶ TaskServiceImpl.java
    ▼ pt.unl.fct.iadi.main.tests
      ▶ HelloControllerTest.java
      ▶ TaskControllerTest.java
      src.pt.unl.fct
  ▶ Maven Dependencies
  ▶ bin
  ▶ target
    application.properties
    pom.xml
```

Where the magic begins (1/2):

The TaskService interface and TaskServiceImpl provide an interface and implementation to access the storage containing tasks.

Spring by example…

spring
by Pivotal



```
▼ SpringExample2
  ▶ JRE System Library [JavaSE-1.8]
  ▼ src
    ▼ pt.unl.fct.iadi.main
      ▶ Application.java
    ▼ pt.unl.fct.iadi.main.controllers
      ▶ HelloController.java
      ▶ Preconditions.java
      ▶ TasksController.java
    ▶ pt.unl.fct.iadi.main.exceptions
    ▼ pt.unl.fct.iadi.main.model
      ▶ Task.java
      ▶ TaskBuilder.java
      ▶ TaskRepository.java
    ▼ pt.unl.fct.iadi.main.services
      ▶ TaskService.java
      ▶ TaskServiceImpl.java
    ▼ pt.unl.fct.iadi.main.tests
      ▶ HelloControllerTest.java
      ▶ TaskControllerTest.java
      src.pt.unl.fct
  ▶ Maven Dependencies
  ▶ bin
  ▶ target
    application.properties
    pom.xml
```

This is going to be used here instead of the awful HashTable in the previous encarnation of the example.

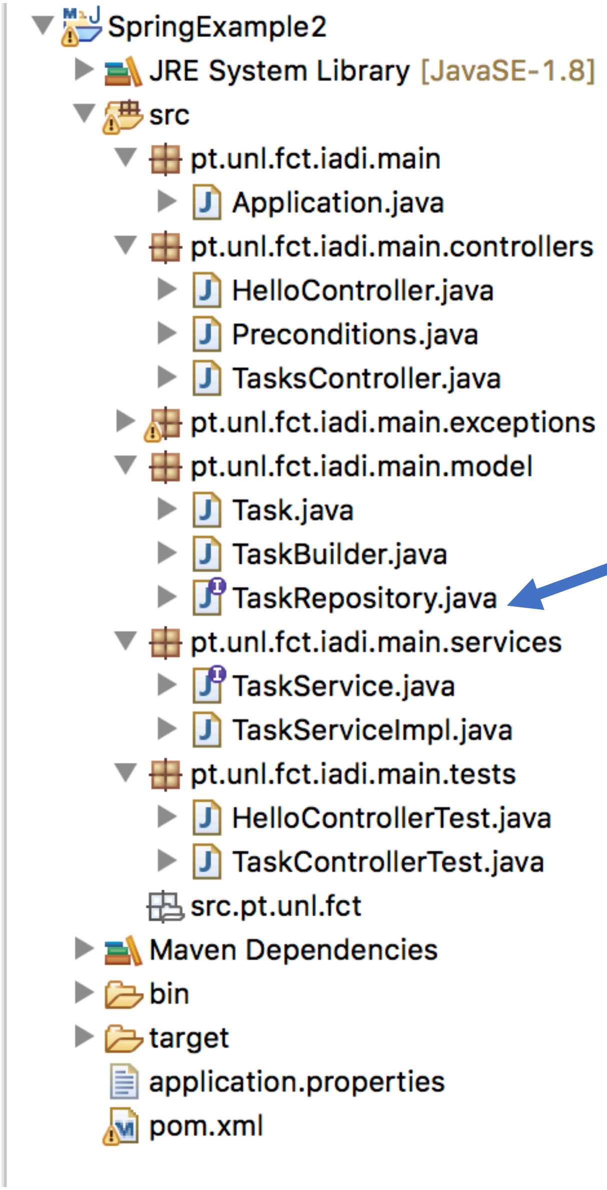
Where the magic begins (1/2):

The TaskService interface and TaskServiceImpl provide an interface and implementation to access the storage containing tasks.

Spring by example…

spring

by Pivotal

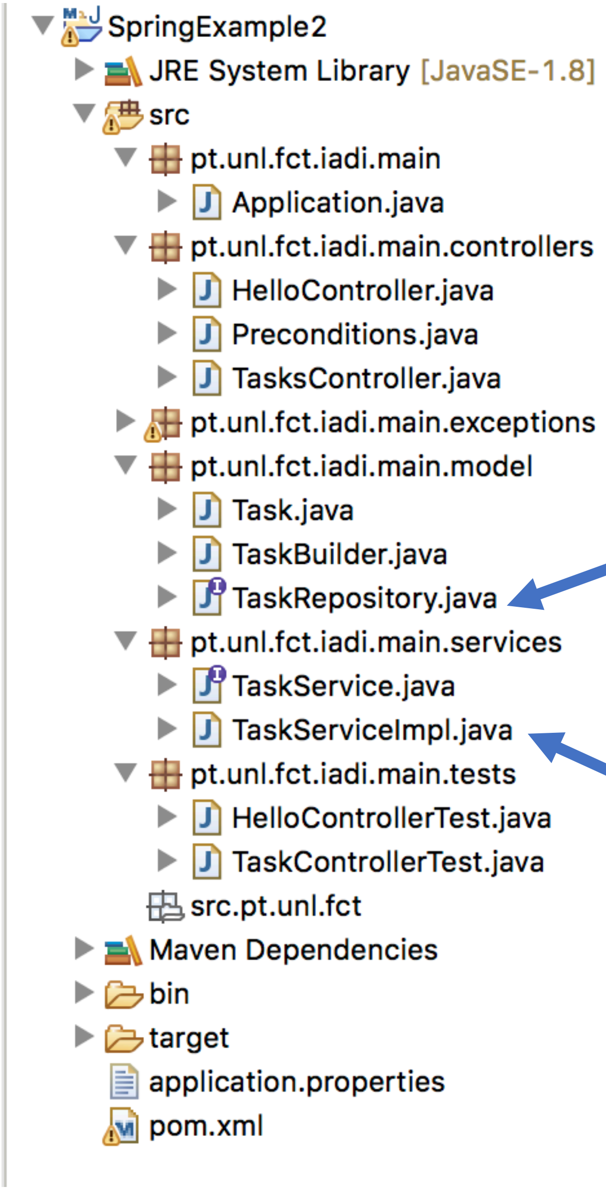


```
▼ SpringExample2
  ▶ JRE System Library [JavaSE-1.8]
  ▼ src
    ▼ pt.unl.fct.iadi.main
      ▶ Application.java
    ▼ pt.unl.fct.iadi.main.controllers
      ▶ HelloController.java
      ▶ Preconditions.java
      ▶ TasksController.java
    ▶ pt.unl.fct.iadi.main.exceptions
    ▼ pt.unl.fct.iadi.main.model
      ▶ Task.java
      ▶ TaskBuilder.java
      ▶ TaskRepository.java
    ▼ pt.unl.fct.iadi.main.services
      ▶ TaskService.java
      ▶ TaskServiceImpl.java
    ▼ pt.unl.fct.iadi.main.tests
      ▶ HelloControllerTest.java
      ▶ TaskControllerTest.java
    src.pt.unl.fct
  ▶ Maven Dependencies
  ▶ bin
  ▶ target
    application.properties
    pom.xml
```

Where the magic begins (2/2):

This interface defines the fundamental mechanisms (i.e, interface) provided by the concrete storage system being used.

Spring by example…



```
▼ SpringExample2
  ▶ JRE System Library [JavaSE-1.8]
  ▼ src
    ▼ pt.unl.fct.iadi.main
      ▶ Application.java
    ▼ pt.unl.fct.iadi.main.controllers
      ▶ HelloController.java
      ▶ Preconditions.java
      ▶ TasksController.java
    ▶ pt.unl.fct.iadi.main.exceptions
    ▼ pt.unl.fct.iadi.main.model
      ▶ Task.java
      ▶ TaskBuilder.java
      ▶ TaskRepository.java
    ▼ pt.unl.fct.iadi.main.services
      ▶ TaskService.java
      ▶ TaskServiceImpl.java
    ▼ pt.unl.fct.iadi.main.tests
      ▶ HelloControllerTest.java
      ▶ TaskControllerTest.java
      src.pt.unl.fct
  ▶ Maven Dependencies
  ▶ bin
  ▶ target
    application.properties
    pom.xml
```

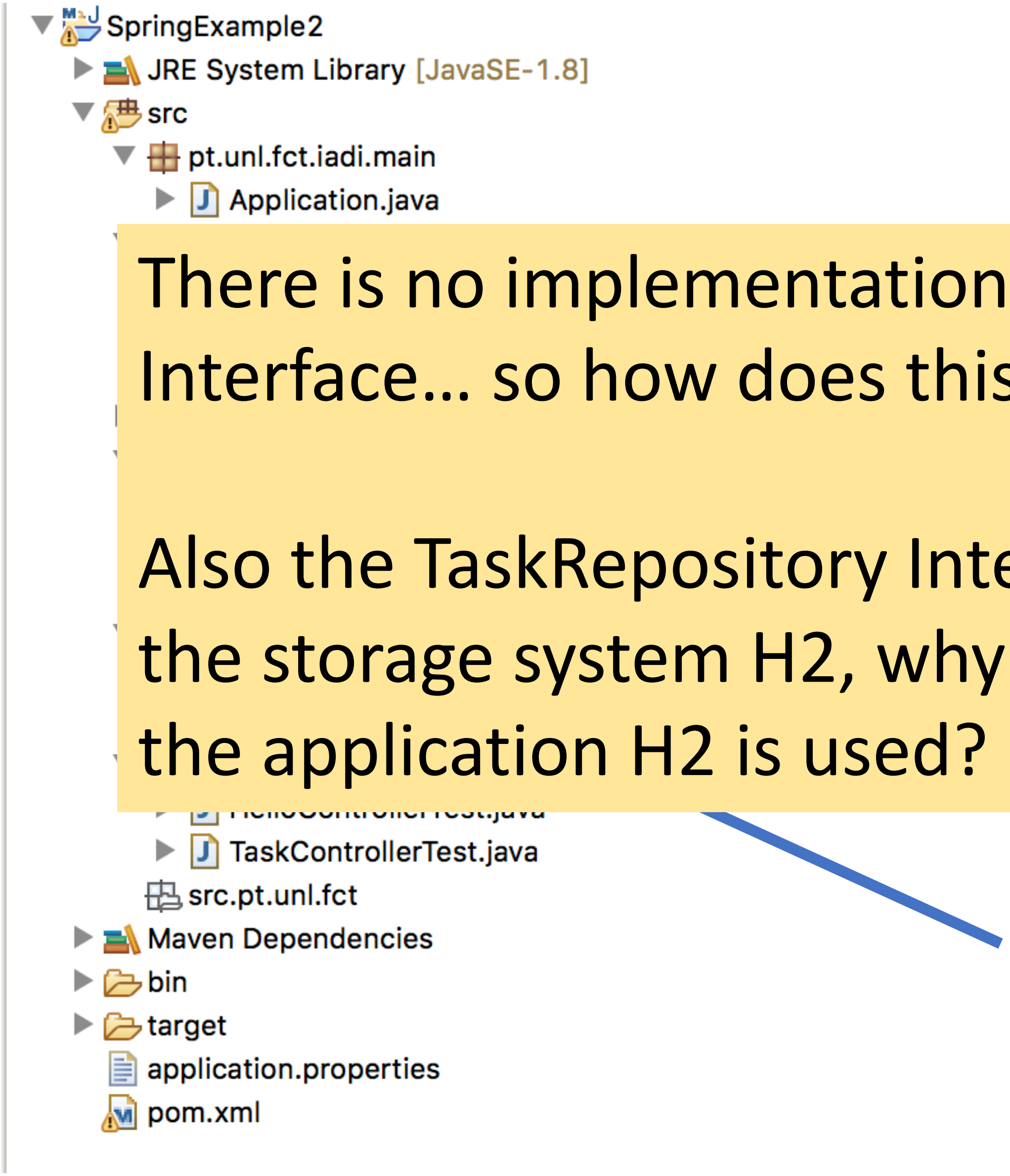
Where the magic begins (2/2):

This interface defines the fundamental mechanisms (i.e, interface) provided by the concrete storage system being used.

This is going to be used by the TaskServiceImpl to store and access data in the storage system.

Spring by example…

Where the magic begins (2/2):



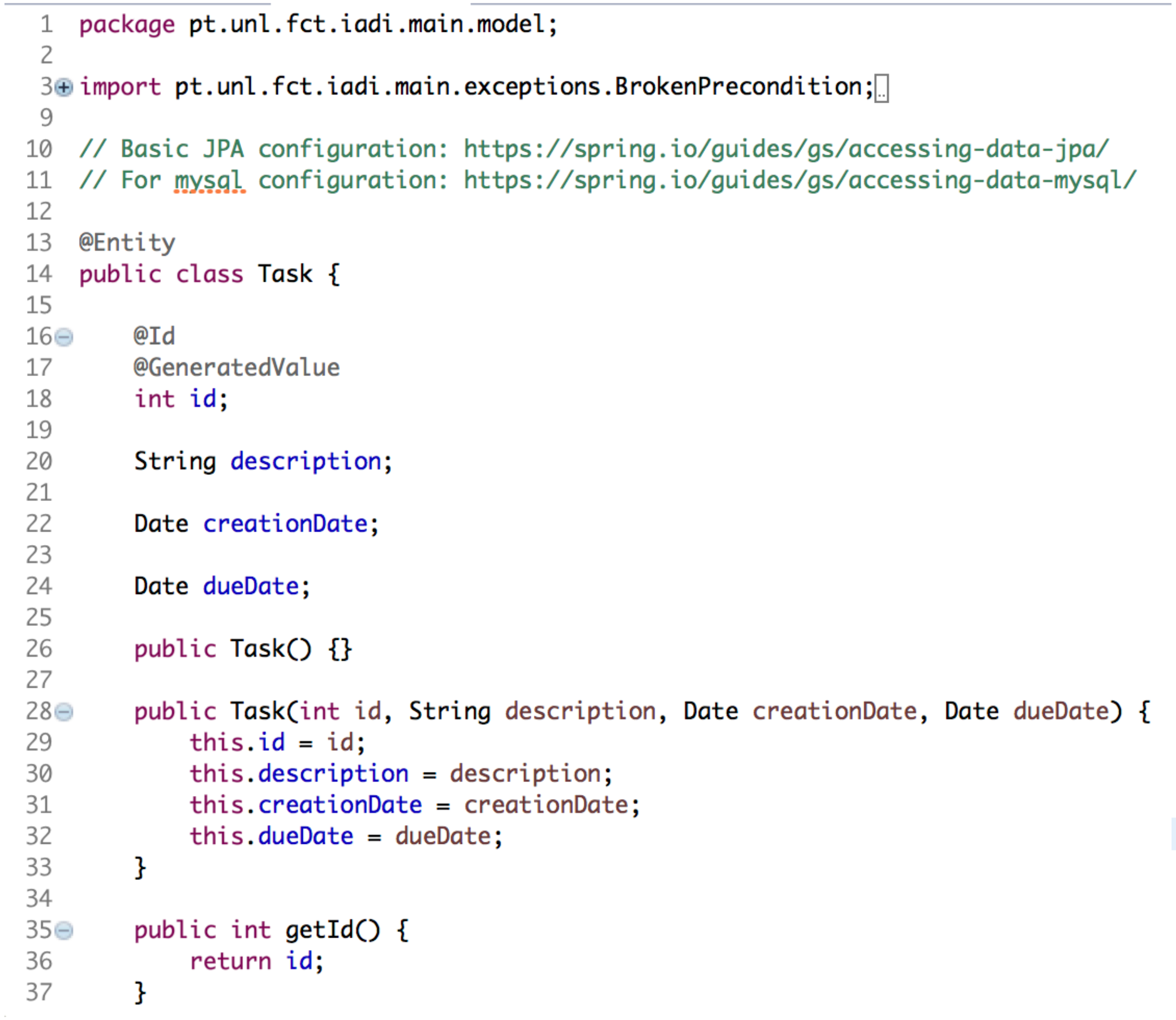
```
SpringExample2
  JRE System Library [JavaSE-1.8]
  src
    pt.unl.fct.iadi.main
      Application.java
      HelloControllerTest.java
      TaskControllerTest.java
    src.pt.unl.fct
  Maven Dependencies
  bin
  target
  application.properties
  pom.xml
```

There is no implementation of this TaskRepository Interface… so how does this work in practice?

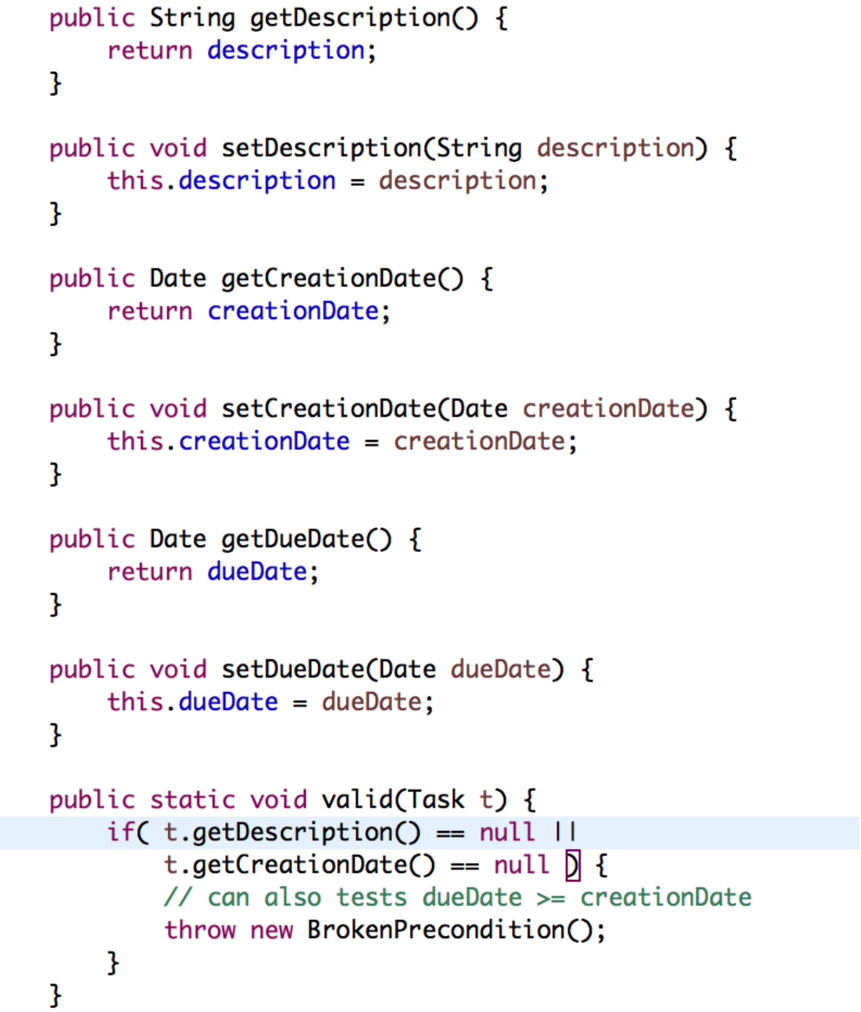
Also the TaskRepository Interface never mentions the storage system H2, why is it that when we run the application H2 is used?

This is going to be used by the TaskServiceImpl to store and access data in the storage system.

Spring by example…

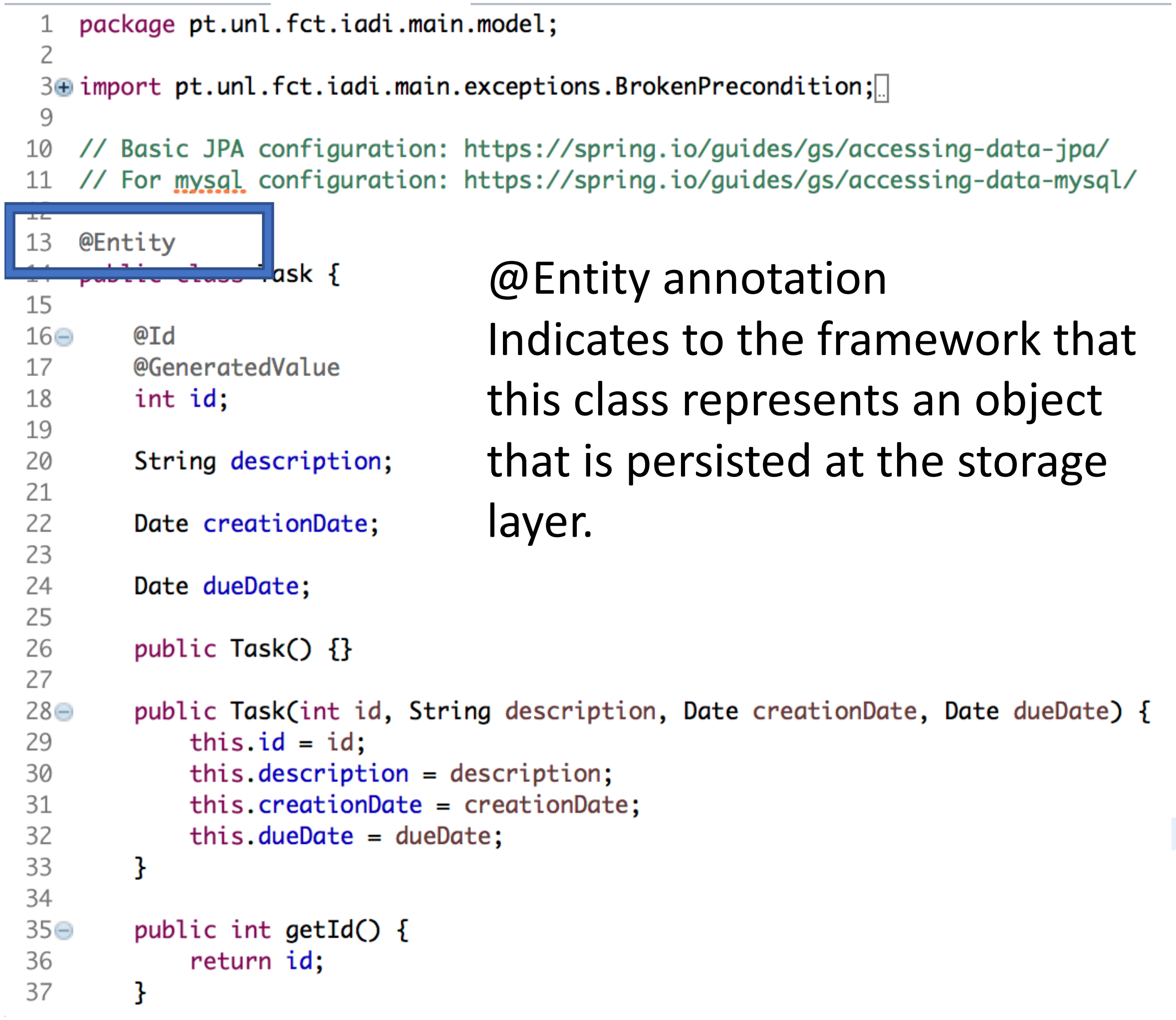


```java
package pt.unl.fct.iadi.main.model;

import pt.unl.fct.iadi.main.exceptions.BrokenPrecondition;

// Basic JPA configuration: https://spring.io/guides/gs/accessing-data-jpa/
// For mysql configuration: https://spring.io/guides/gs/accessing-data-mysql/

@Entity
public class Task {

    @Id
    @GeneratedValue
    int id;

    String description;

    Date creationDate;

    Date dueDate;

    public Task() {}

    public Task(int id, String description, Date creationDate, Date dueDate) {
        this.id = id;
        this.description = description;
        this.creationDate = creationDate;
        this.dueDate = dueDate;
    }

    public int getId() {
        return id;
    }
```



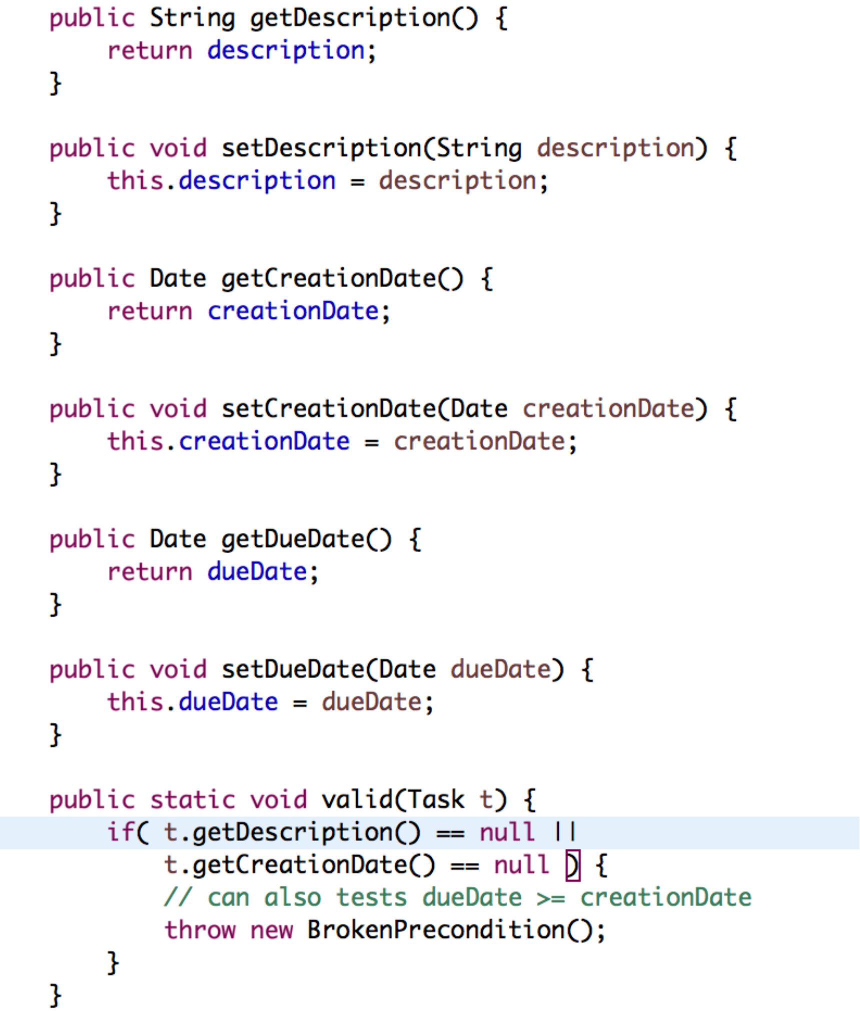
```java
public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

public Date getCreationDate() {
    return creationDate;
}

public void setCreationDate(Date creationDate) {
    this.creationDate = creationDate;
}

public Date getDueDate() {
    return dueDate;
}

public void setDueDate(Date dueDate) {
    this.dueDate = dueDate;
}

public static void valid(Task t) {
    if( t.getDescription() == null ||
        t.getCreationDate() == null ) {
        // can also tests dueDate >= creationDate
        throw new BrokenPrecondition();
    }
}
```

Spring by example…



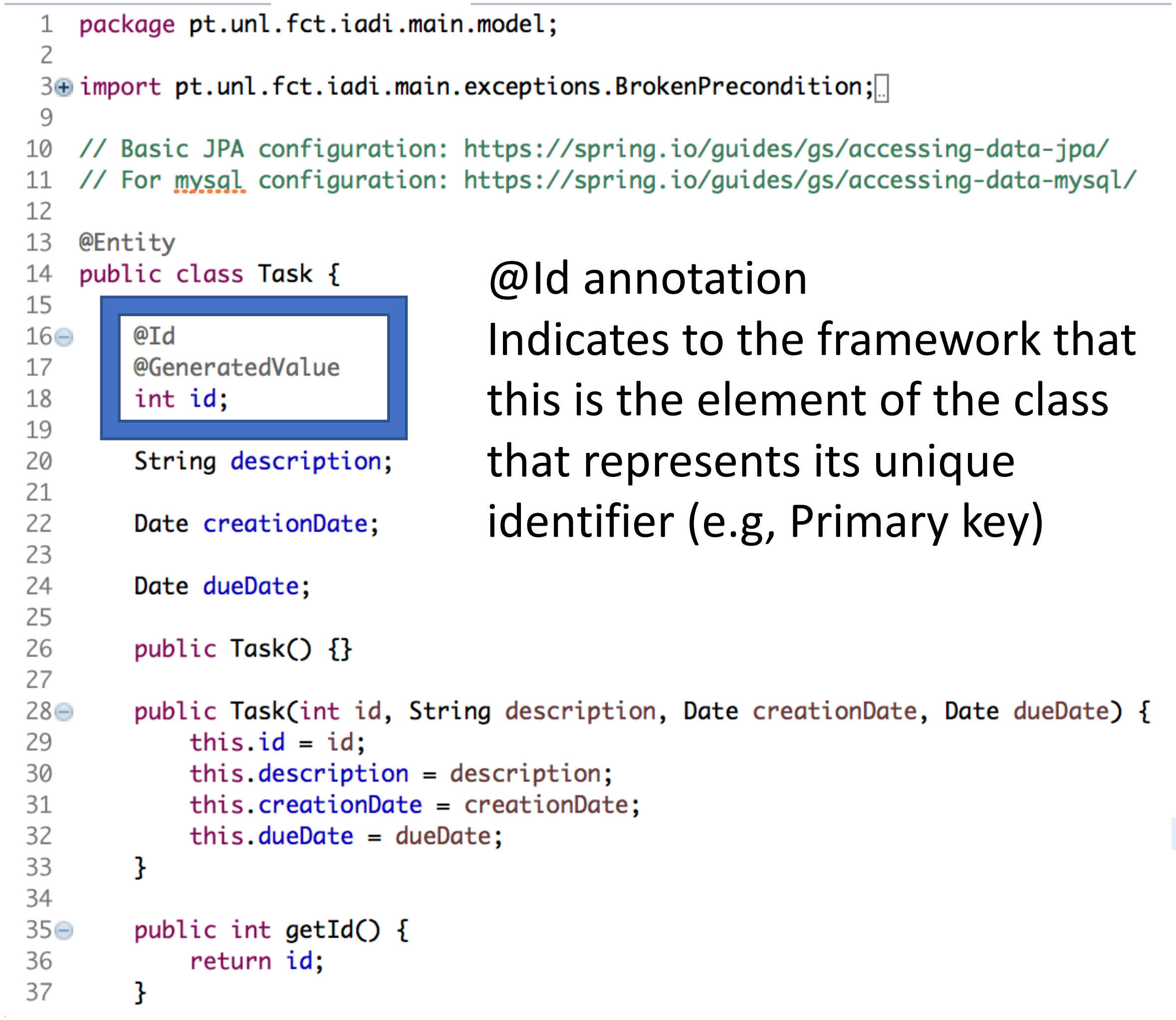
```java
package pt.unl.fct.iadi.main.model;

import pt.unl.fct.iadi.main.exceptions.BrokenPrecondition;

// Basic JPA configuration: https://spring.io/guides/gs/accessing-data-jpa/
// For mysql configuration: https://spring.io/guides/gs/accessing-data-mysql/

@Entity
public class Task {

    @Id
    @GeneratedValue
    int id;

    String description;

    Date creationDate;

    Date dueDate;

    public Task() {}

    public Task(int id, String description, Date creationDate, Date dueDate) {
        this.id = id;
        this.description = description;
        this.creationDate = creationDate;
        this.dueDate = dueDate;
    }

    public int getId() {
        return id;
    }
```

@Entity annotation
Indicates to the framework that this class represents an object that is persisted at the storage layer.



```java
public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

public Date getCreationDate() {
    return creationDate;
}

public void setCreationDate(Date creationDate) {
    this.creationDate = creationDate;
}

public Date getDueDate() {
    return dueDate;
}

public void setDueDate(Date dueDate) {
    this.dueDate = dueDate;
}

public static void valid(Task t) {
    if( t.getDescription() == null ||
        t.getCreationDate() == null ) {
        // can also tests dueDate >= creationDate
        throw new BrokenPrecondition();
    }
}
```

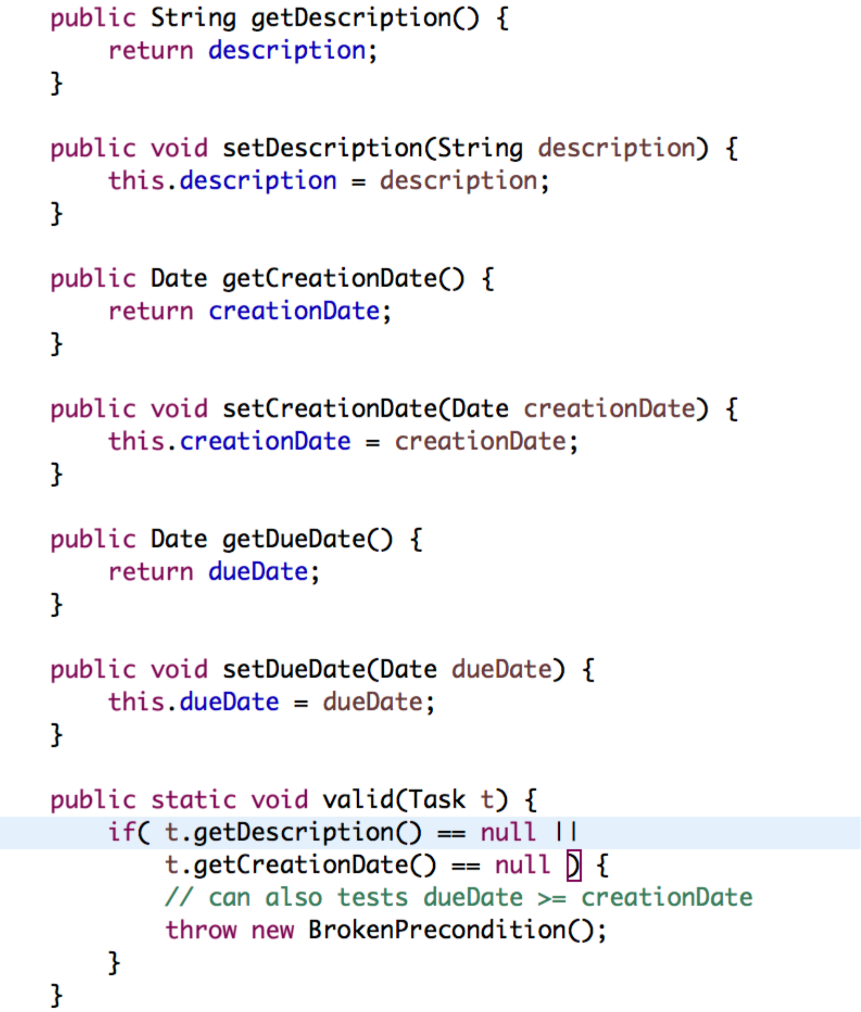
Spring by example…



```java
package pt.unl.fct.iadi.main.model;

import pt.unl.fct.iadi.main.exceptions.BrokenPrecondition;

// Basic JPA configuration: https://spring.io/guides/gs/accessing-data-jpa/
// For mysql configuration: https://spring.io/guides/gs/accessing-data-mysql/

@Entity
public class Task {

    @Id
    @GeneratedValue
    int id;

    String description;

    Date creationDate;

    Date dueDate;

    public Task() {}

    public Task(int id, String description, Date creationDate, Date dueDate) {
        this.id = id;
        this.description = description;
        this.creationDate = creationDate;
        this.dueDate = dueDate;
    }

    public int getId() {
        return id;
    }
```

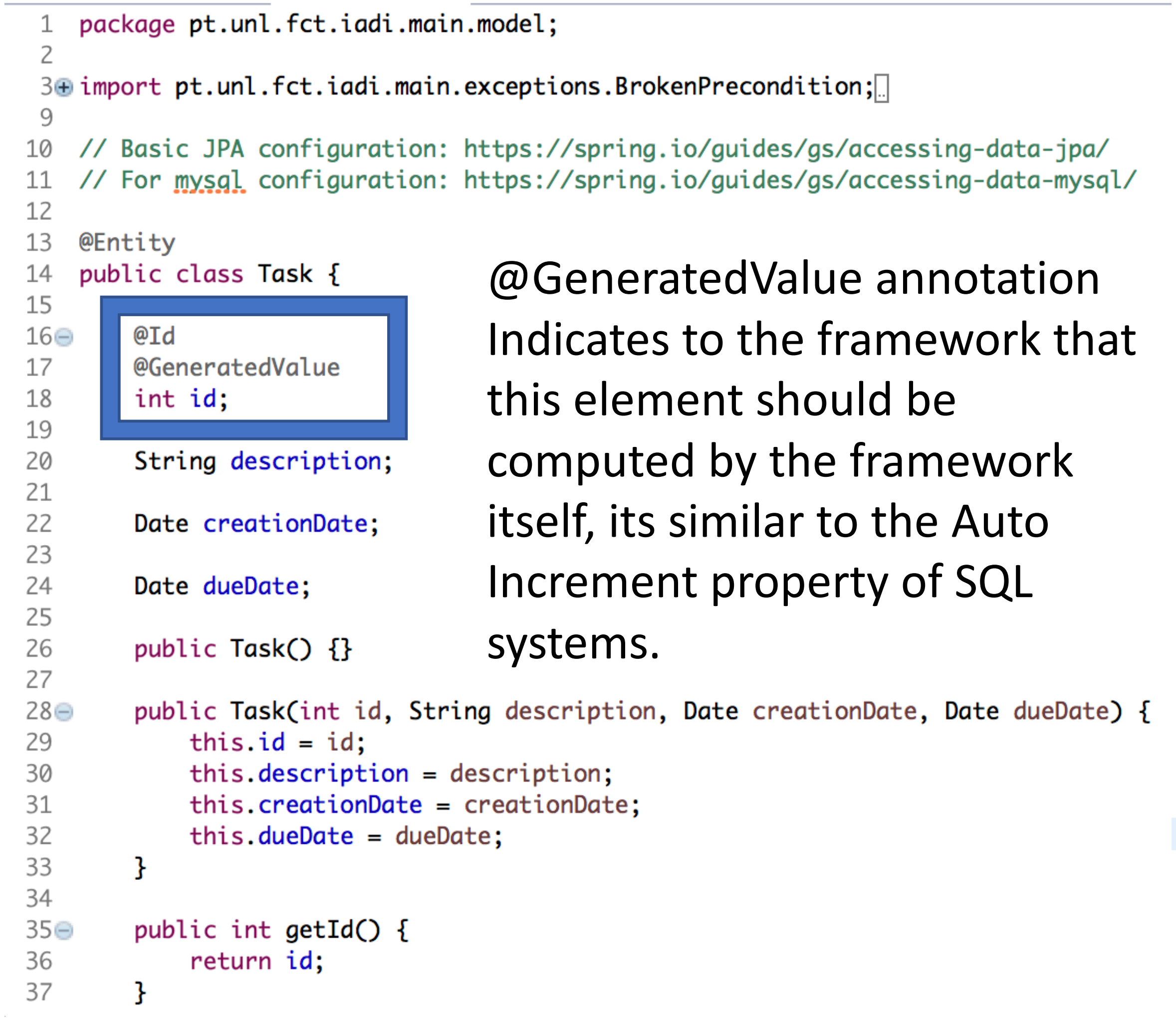
@Id annotation

Indicates to the framework that this is the element of the class that represents its unique identifier (e.g, Primary key)



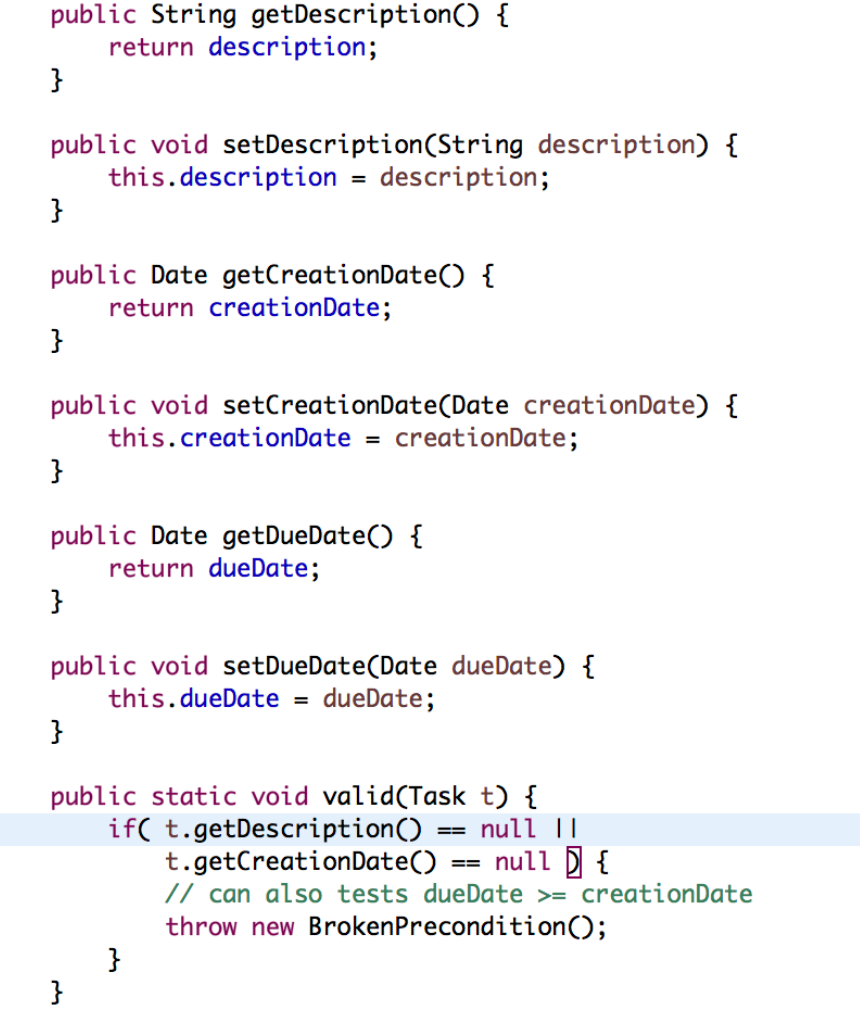
```java
public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

public Date getCreationDate() {
    return creationDate;
}

public void setCreationDate(Date creationDate) {
    this.creationDate = creationDate;
}

public Date getDueDate() {
    return dueDate;
}

public void setDueDate(Date dueDate) {
    this.dueDate = dueDate;
}

public static void valid(Task t) {
    if( t.getDescription() == null ||
        t.getCreationDate() == null ) {
        // can also tests dueDate >= creationDate
        throw new BrokenPrecondition();
    }
}
```

Spring by example…



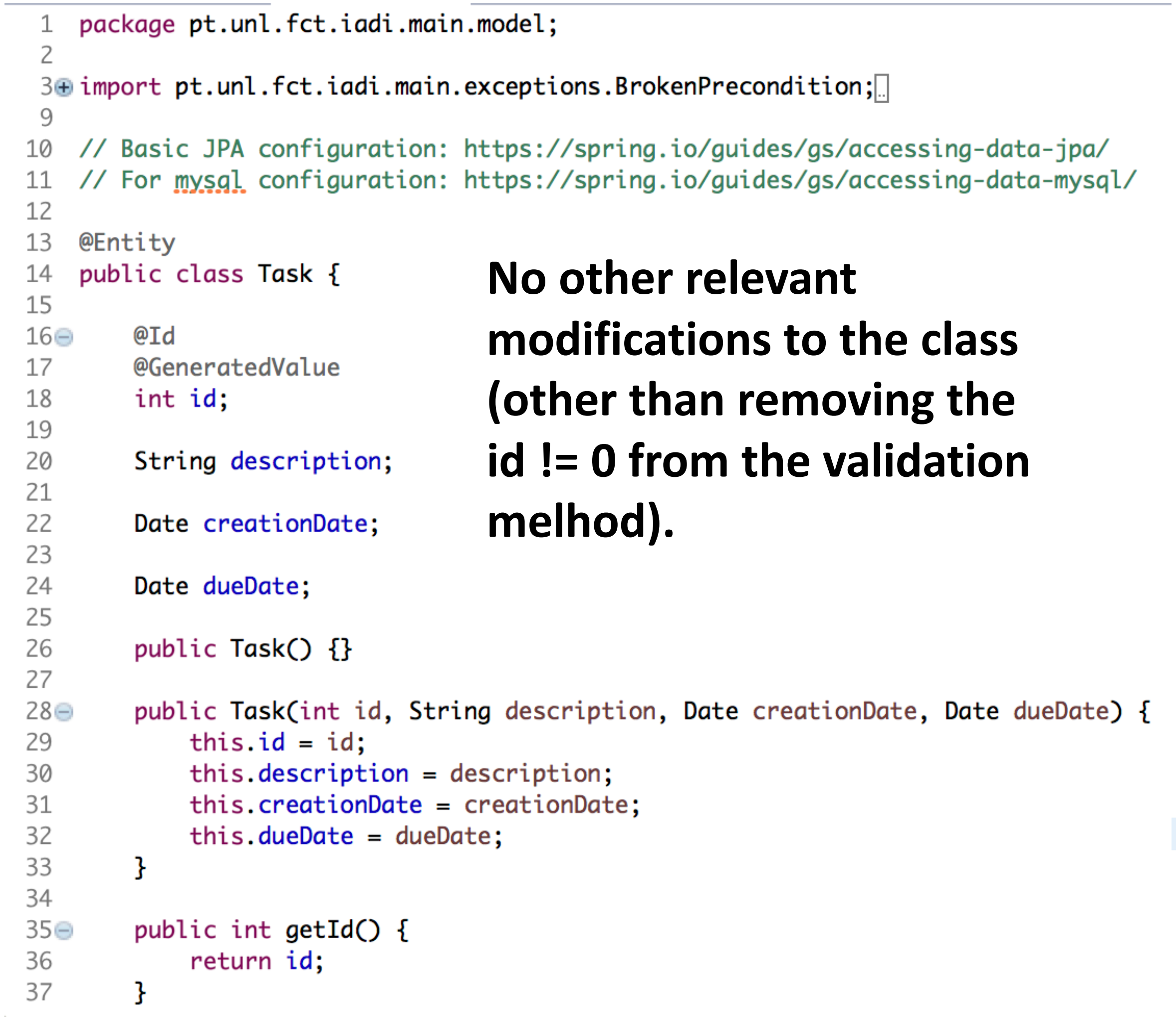
```
package pt.unl.fct.iadi.main.model;

import pt.unl.fct.iadi.main.exceptions.BrokenPrecondition;

// Basic JPA configuration: https://spring.io/guides/gs/accessing-data-jpa/
// For mysql configuration: https://spring.io/guides/gs/accessing-data-mysql/

@Entity
public class Task {

    @Id
    @GeneratedValue
    int id;

    String description;

    Date creationDate;

    Date dueDate;

    public Task() {}

    public Task(int id, String description, Date creationDate, Date dueDate) {
        this.id = id;
        this.description = description;
        this.creationDate = creationDate;
        this.dueDate = dueDate;
    }

    public int getId() {
        return id;
    }
```

@GeneratedValue annotation Indicates to the framework that this element should be computed by the framework itself, its similar to the Auto Increment property of SQL systems.



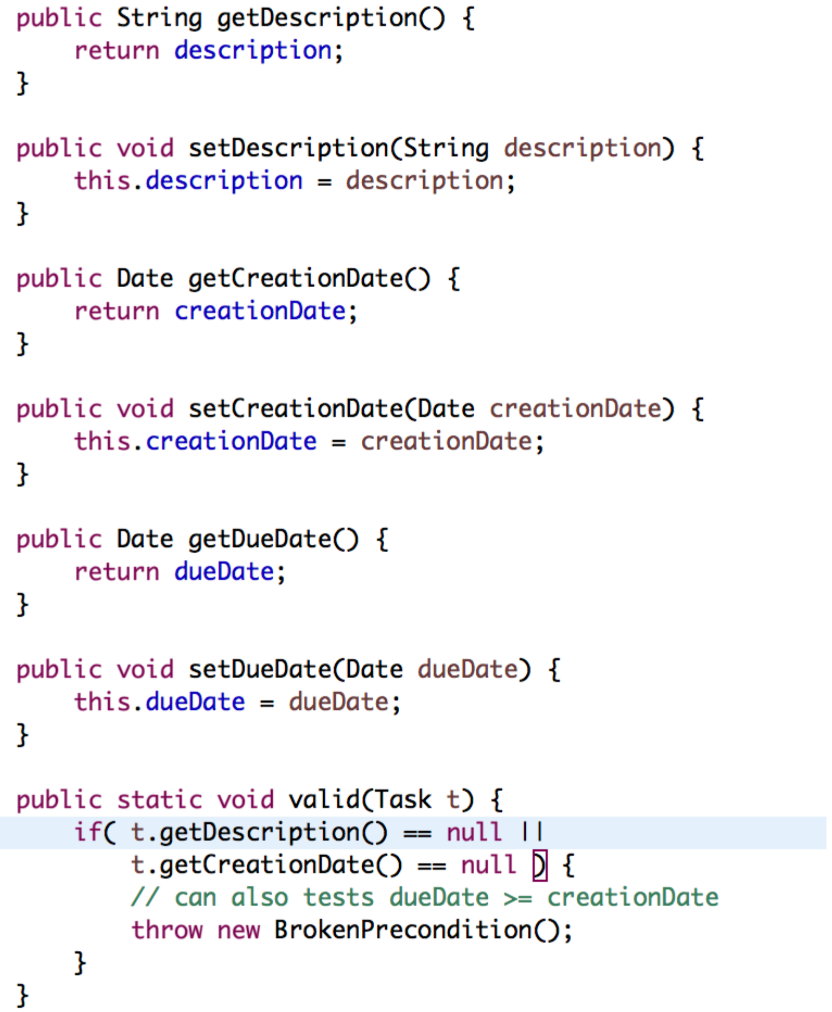
```
public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

public Date getCreationDate() {
    return creationDate;
}

public void setCreationDate(Date creationDate) {
    this.creationDate = creationDate;
}

public Date getDueDate() {
    return dueDate;
}

public void setDueDate(Date dueDate) {
    this.dueDate = dueDate;
}

public static void valid(Task t) {
    if( t.getDescription() == null ||
        t.getCreationDate() == null ) {
        // can also tests dueDate >= creationDate
        throw new BrokenPrecondition();
    }
}
```

Spring by example…



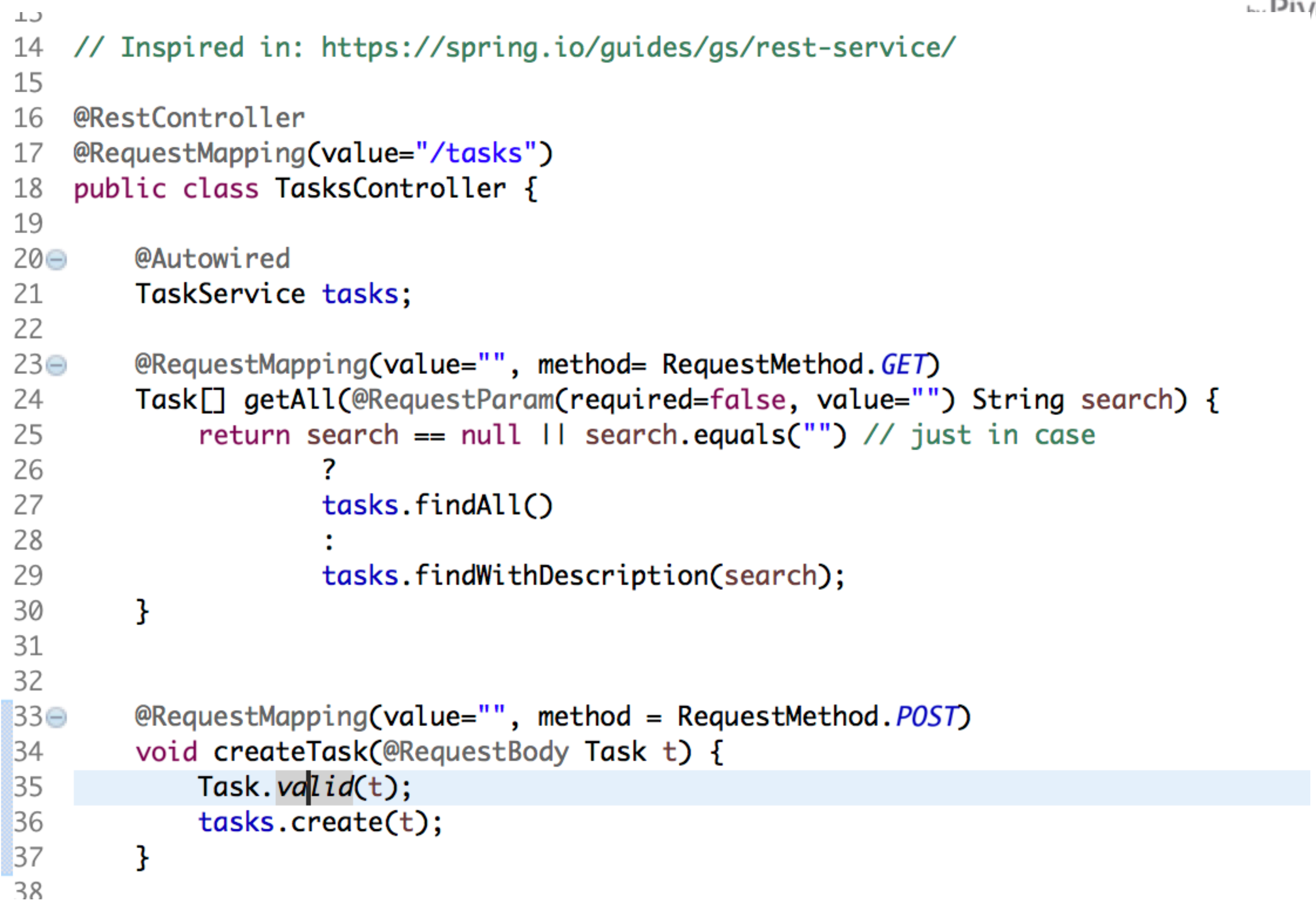
```java
package pt.unl.fct.iadi.main.model;

import pt.unl.fct.iadi.main.exceptions.BrokenPrecondition;

// Basic JPA configuration: https://spring.io/guides/gs/accessing-data-jpa/
// For mysql configuration: https://spring.io/guides/gs/accessing-data-mysql/

@Entity
public class Task {

    @Id
    @GeneratedValue
    int id;

    String description;

    Date creationDate;

    Date dueDate;

    public Task() {}

    public Task(int id, String description, Date creationDate, Date dueDate) {
        this.id = id;
        this.description = description;
        this.creationDate = creationDate;
        this.dueDate = dueDate;
    }

    public int getId() {
        return id;
    }
```

No other relevant modifications to the class (other than removing the id != 0 from the validation melhod).



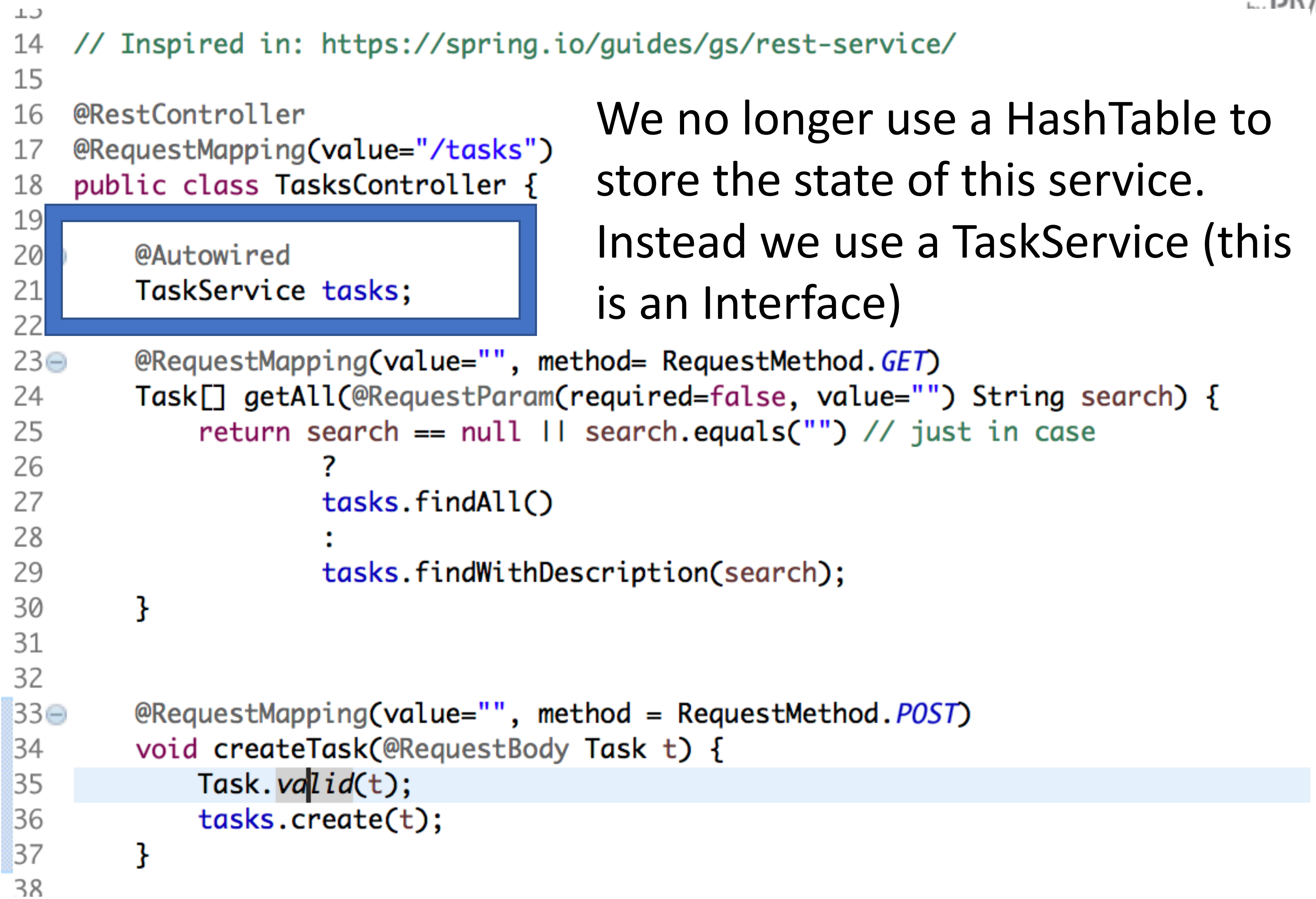
```java
public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

public Date getCreationDate() {
    return creationDate;
}

public void setCreationDate(Date creationDate) {
    this.creationDate = creationDate;
}

public Date getDueDate() {
    return dueDate;
}

public void setDueDate(Date dueDate) {
    this.dueDate = dueDate;
}

public static void valid(Task t) {
    if( t.getDescription() == null ||
        t.getCreationDate() == null ) {
        // can also tests dueDate >= creationDate
        throw new BrokenPrecondition();
    }
}
```

Spring by example…



```java
// Inspired in: https://spring.io/guides/gs/rest-service/

@RestController
@RequestMapping(value="/tasks")
public class TasksController {

    @Autowired
    TaskService tasks;

    @RequestMapping(value="", method= RequestMethod.GET)
    Task[] getAll(@RequestParam(required=false, value="") String search) {
        return search == null || search.equals("") // just in case
                ?
                tasks.findAll()
                :
                tasks.findWithDescription(search);
    }


    @RequestMapping(value="", method = RequestMethod.POST)
    void createTask(@RequestBody Task t) {
        Task.valid(t);
        tasks.create(t);
    }
```

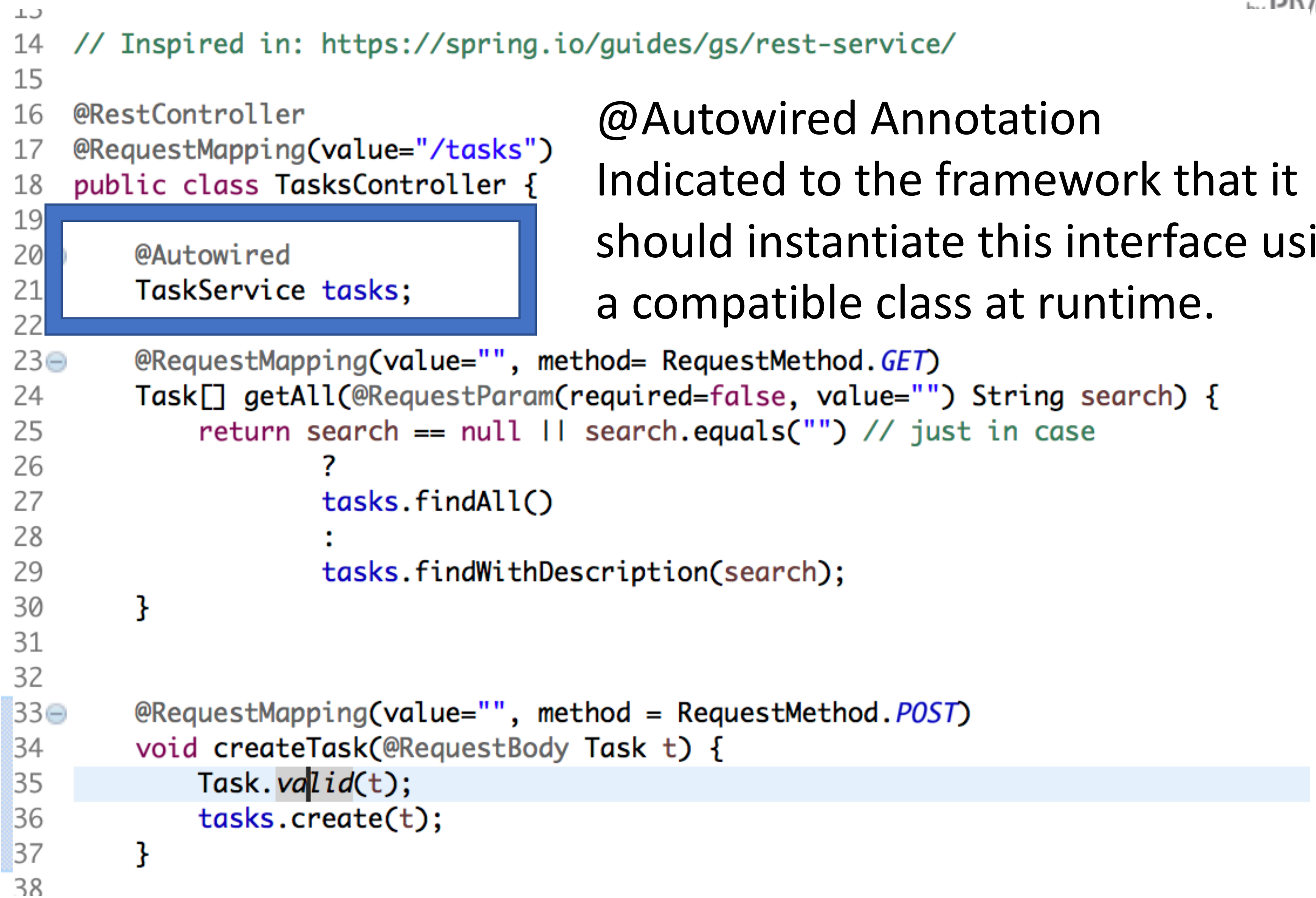
Spring by example…



```java
// Inspired in: https://spring.io/guides/gs/rest-service/

@RestController
@RequestMapping(value="/tasks")
public class TasksController {

    @Autowired
    TaskService tasks;

    @RequestMapping(value="", method= RequestMethod.GET)
    Task[] getAll(@RequestParam(required=false, value="") String search) {
        return search == null || search.equals("") // just in case
                ?
                tasks.findAll()
                :
                tasks.findWithDescription(search);
    }


    @RequestMapping(value="", method = RequestMethod.POST)
    void createTask(@RequestBody Task t) {
        Task.valid(t);
        tasks.create(t);
    }
```

We no longer use a HashTable to store the state of this service. Instead we use a TaskService (this is an Interface)

Spring by example…

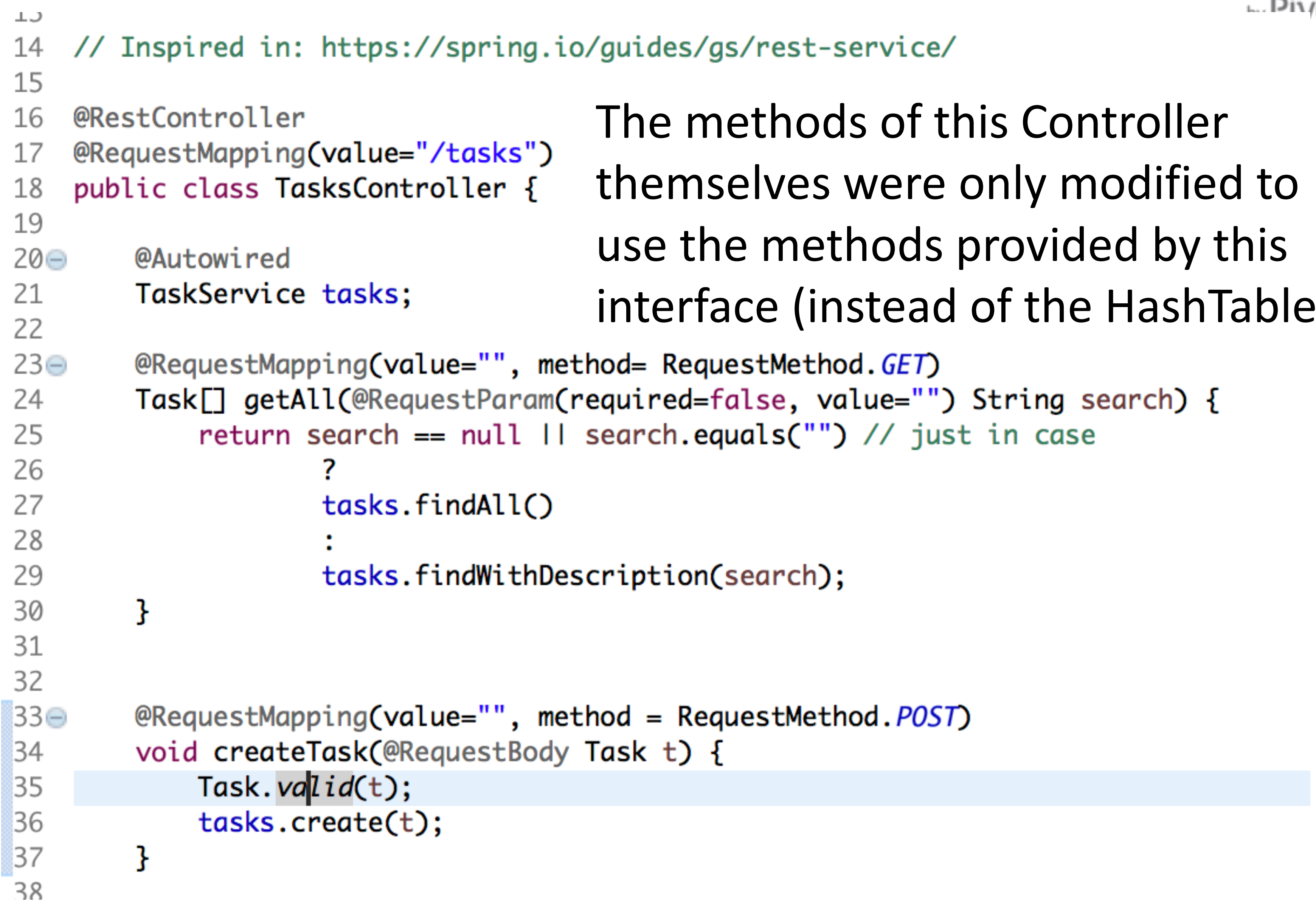


```
// Inspired in: https://spring.io/guides/gs/rest-service/

@RestController
@RequestMapping(value="/tasks")
public class TasksController {

    @Autowired
    TaskService tasks;

    @RequestMapping(value="", method= RequestMethod.GET)
    Task[] getAll(@RequestParam(required=false, value="") String search) {
        return search == null || search.equals("") // just in case
                ?
                tasks.findAll()
                :
                tasks.findWithDescription(search);
    }


    @RequestMapping(value="", method = RequestMethod.POST)
    void createTask(@RequestBody Task t) {
        Task.valid(t);
        tasks.create(t);
    }
```

@Autowired Annotation

Indicated to the framework that it should instantiate this interface using a compatible class at runtime.

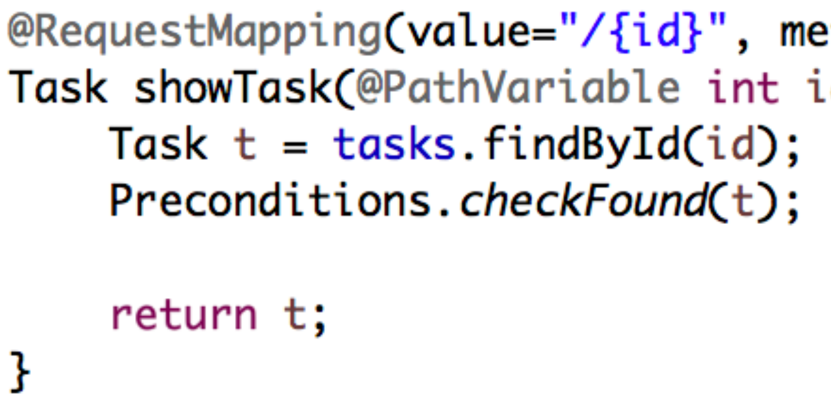
Spring by example…



```java
// Inspired in: https://spring.io/guides/gs/rest-service/

@RestController
@RequestMapping(value="/tasks")
public class TasksController {

    @Autowired
    TaskService tasks;

    @RequestMapping(value="", method= RequestMethod.GET)
    Task[] getAll(@RequestParam(required=false, value="") String search) {
        return search == null || search.equals("") // just in case
                ?
                tasks.findAll()
                :
                tasks.findWithDescription(search);
    }


    @RequestMapping(value="", method = RequestMethod.POST)
    void createTask(@RequestBody Task t) {
        Task.valid(t);
        tasks.create(t);
    }
```

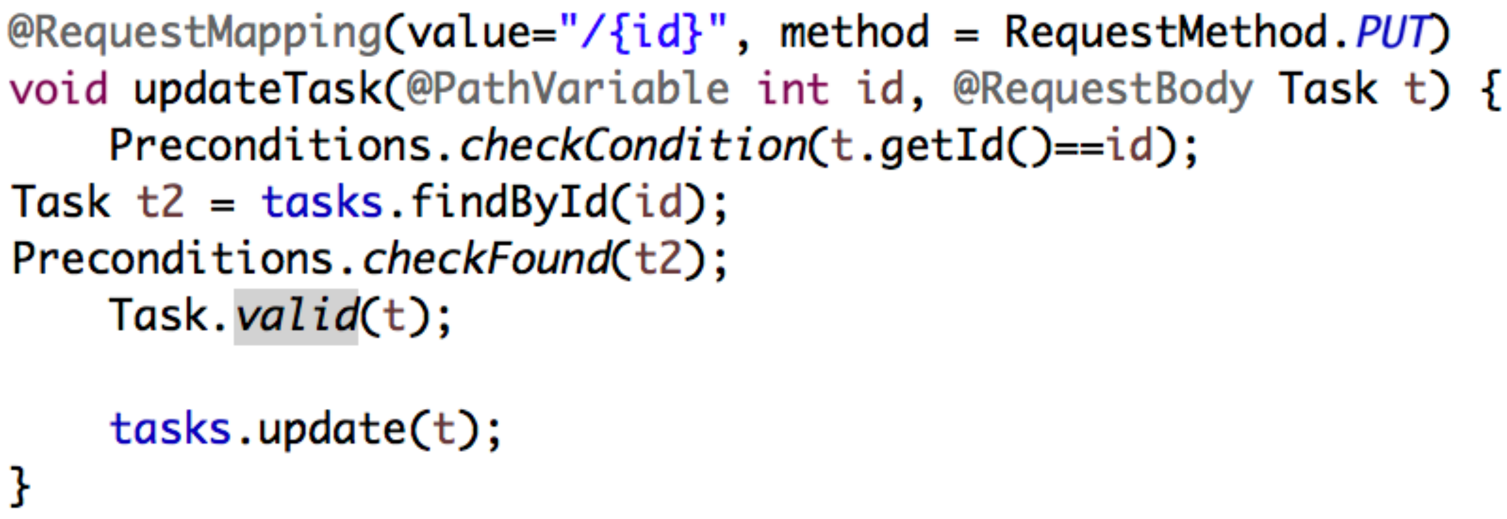
The methods of this Controller themselves were only modified to use the methods provided by this interface (instead of the HashTable)

Spring by example…



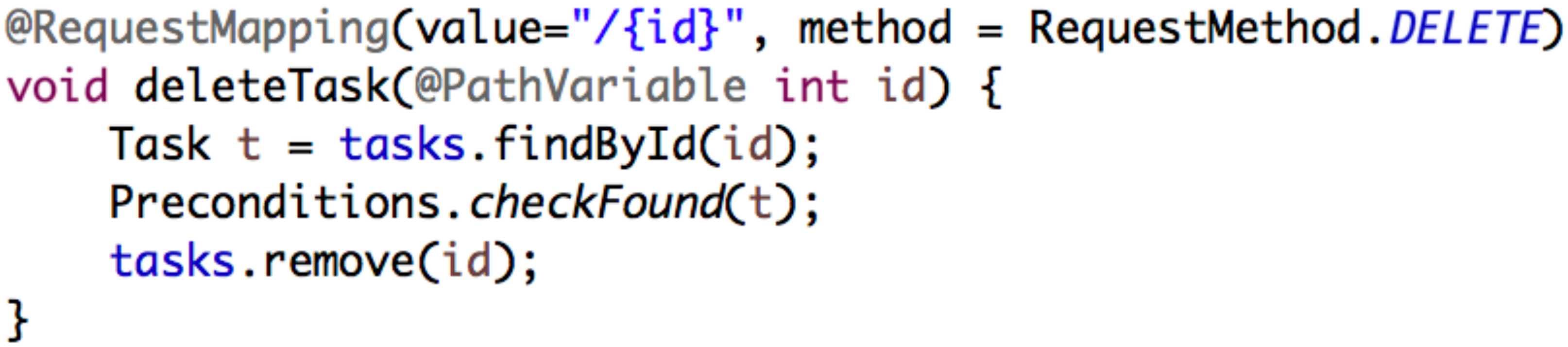
```
@RequestMapping(value="/{id}", method = RequestMethod.GET)
Task showTask(@PathVariable int id) {
    Task t = tasks.findById(id);
    Preconditions.checkFound(t);

    return t;
}
```

The methods of this Controller themselves were only modified to use the methods provided by this interface (instead of the HashTable)



```
@RequestMapping(value="/{id}", method = RequestMethod.PUT)
void updateTask(@PathVariable int id, @RequestBody Task t) {
    Preconditions.checkCondition(t.getId()==id);
Task t2 = tasks.findById(id);
Preconditions.checkFound(t2);
    Task.valid(t);

    tasks.update(t);
}
```

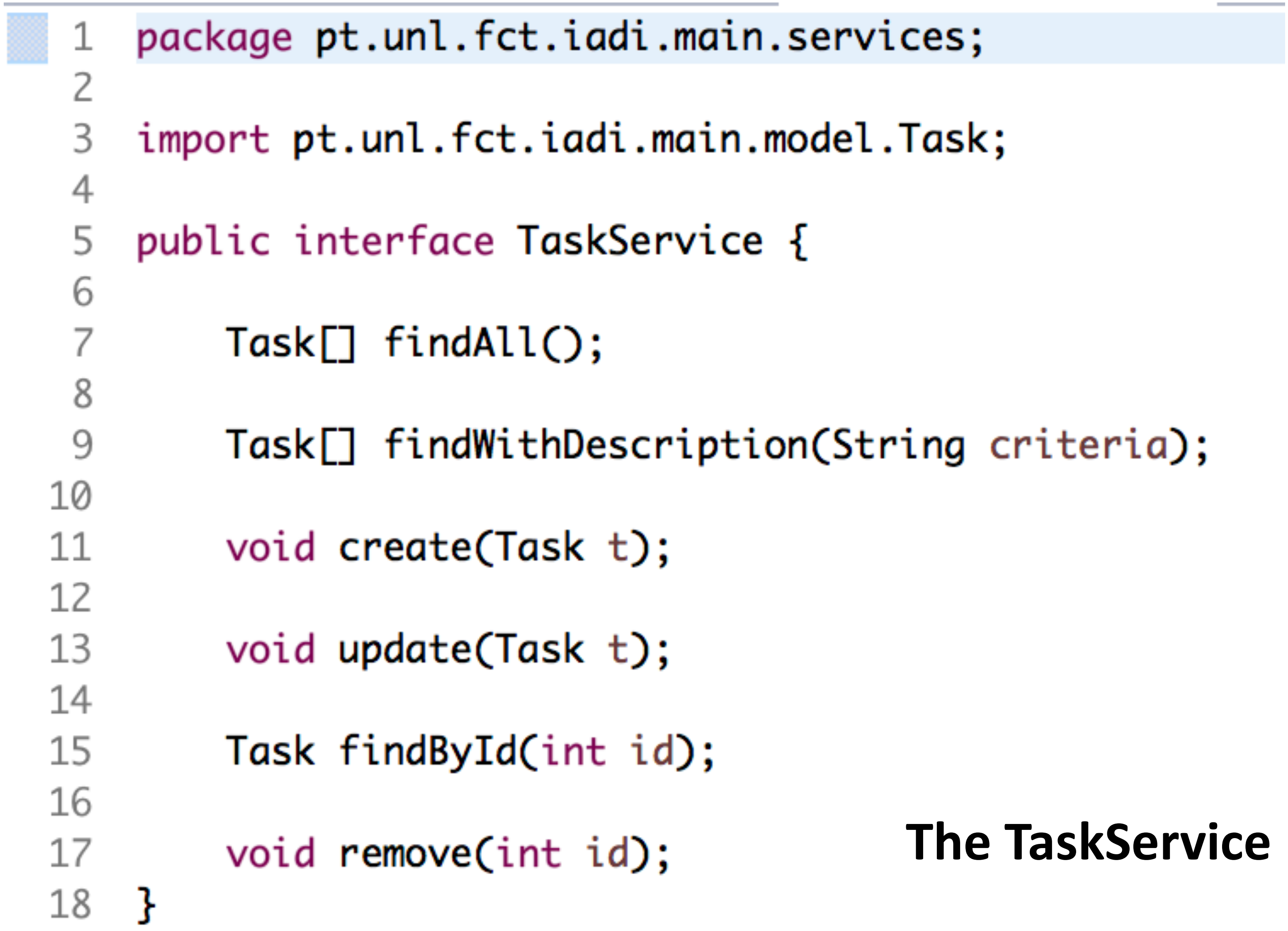
Spring by example…



```
@RequestMapping(value="/{id}", method = RequestMethod.DELETE)
void deleteTask(@PathVariable int id) {
    Task t = tasks.findById(id);
    Preconditions.checkFound(t);
    tasks.remove(id);
}
```

The methods of this Controller themselves were only modified to use the methods provided by this interface (instead of the HashTable)

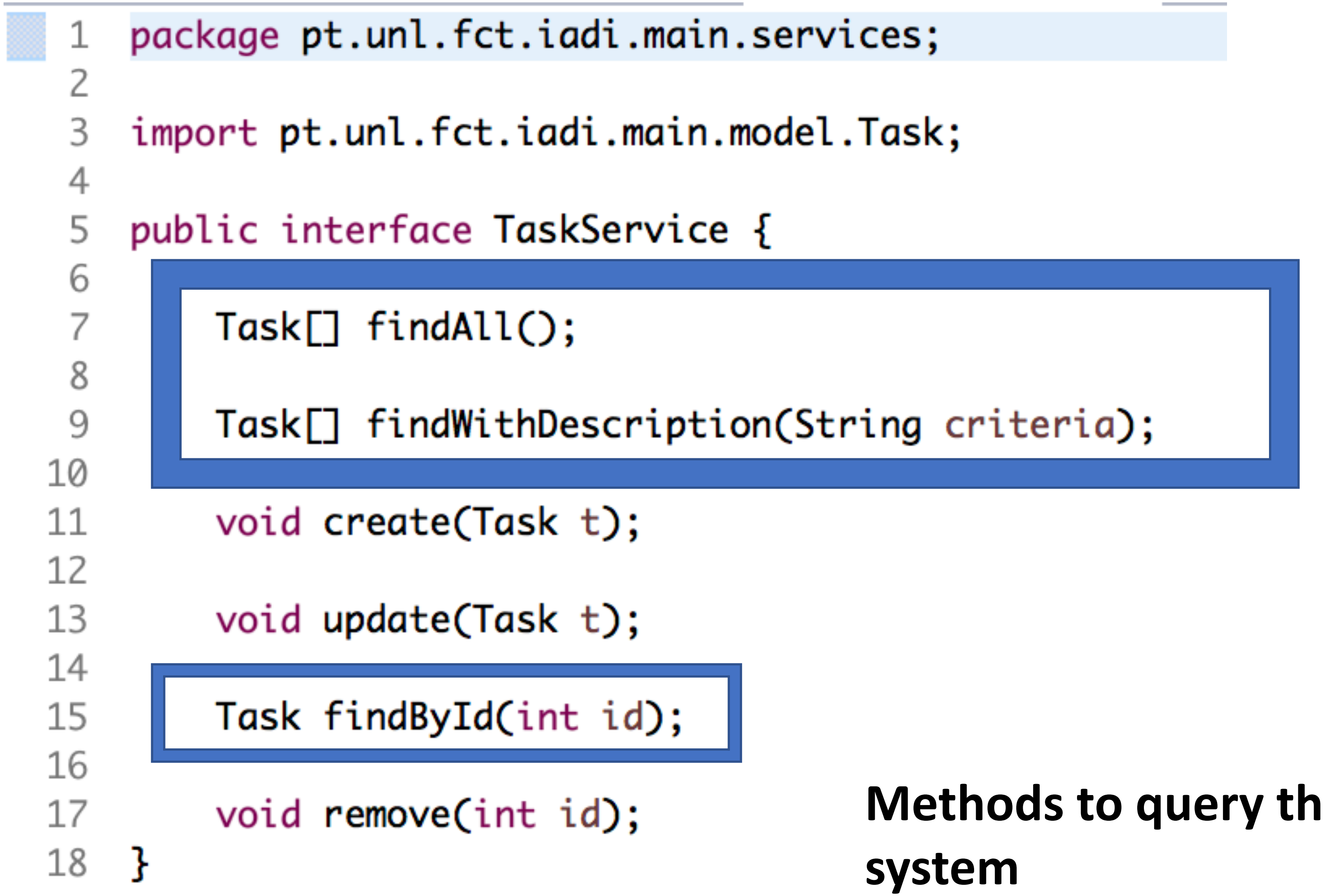
Spring by example...



```java
package pt.unl.fct.iadi.main.services;

import pt.unl.fct.iadi.main.model.Task;

public interface TaskService {

    Task[] findAll();

    Task[] findWithDescription(String criteria);

    void create(Task t);

    void update(Task t);

    Task findById(int id);

    void remove(int id);
}
```

The TaskService Interface.

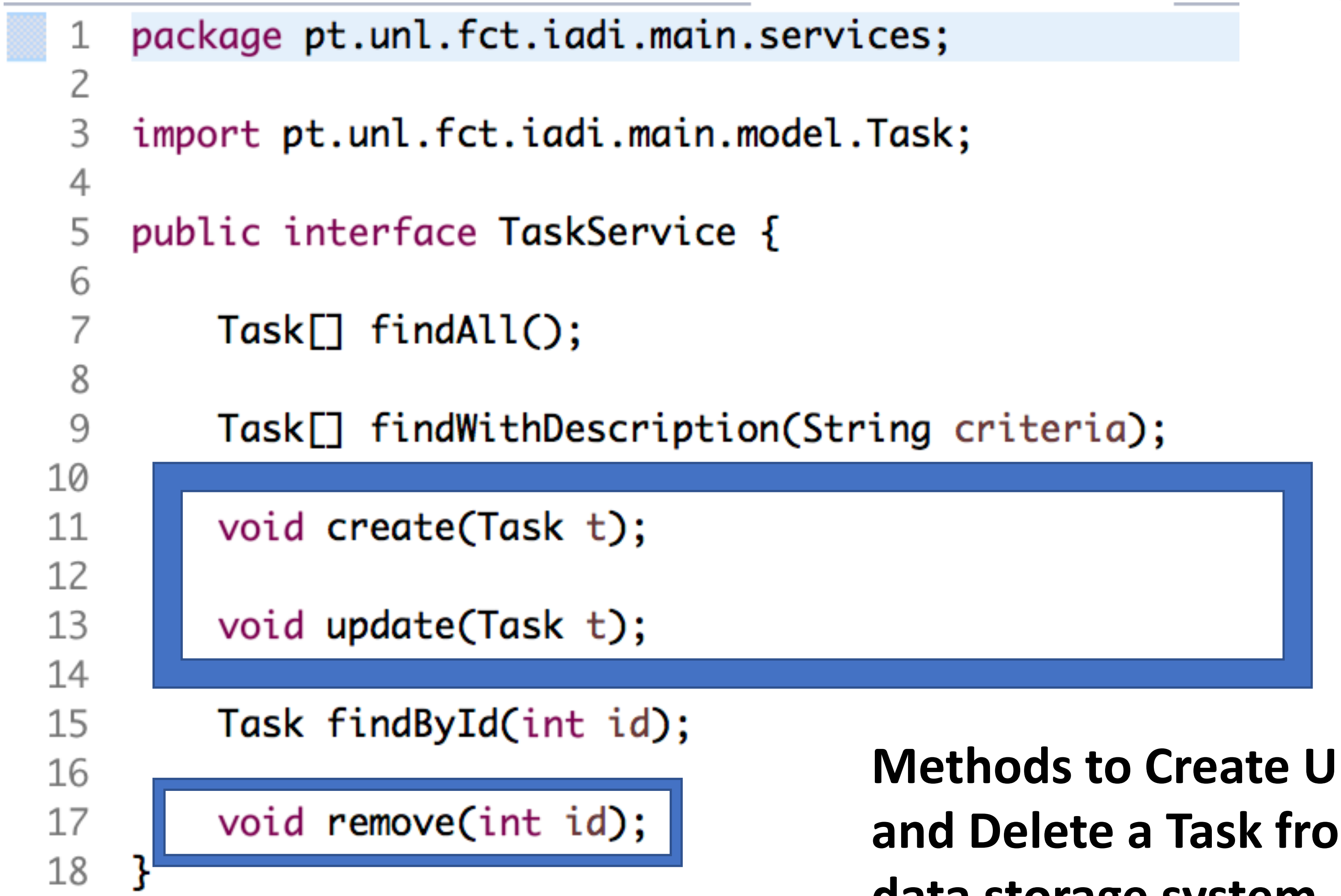
Spring by example...



```java
package pt.unl.fct.iadi.main.services;

import pt.unl.fct.iadi.main.model.Task;

public interface TaskService {

    Task[] findAll();

    Task[] findWithDescription(String criteria);

    void create(Task t);

    void update(Task t);

    Task findById(int id);

    void remove(int id);
}
```

Methods to query the storage system

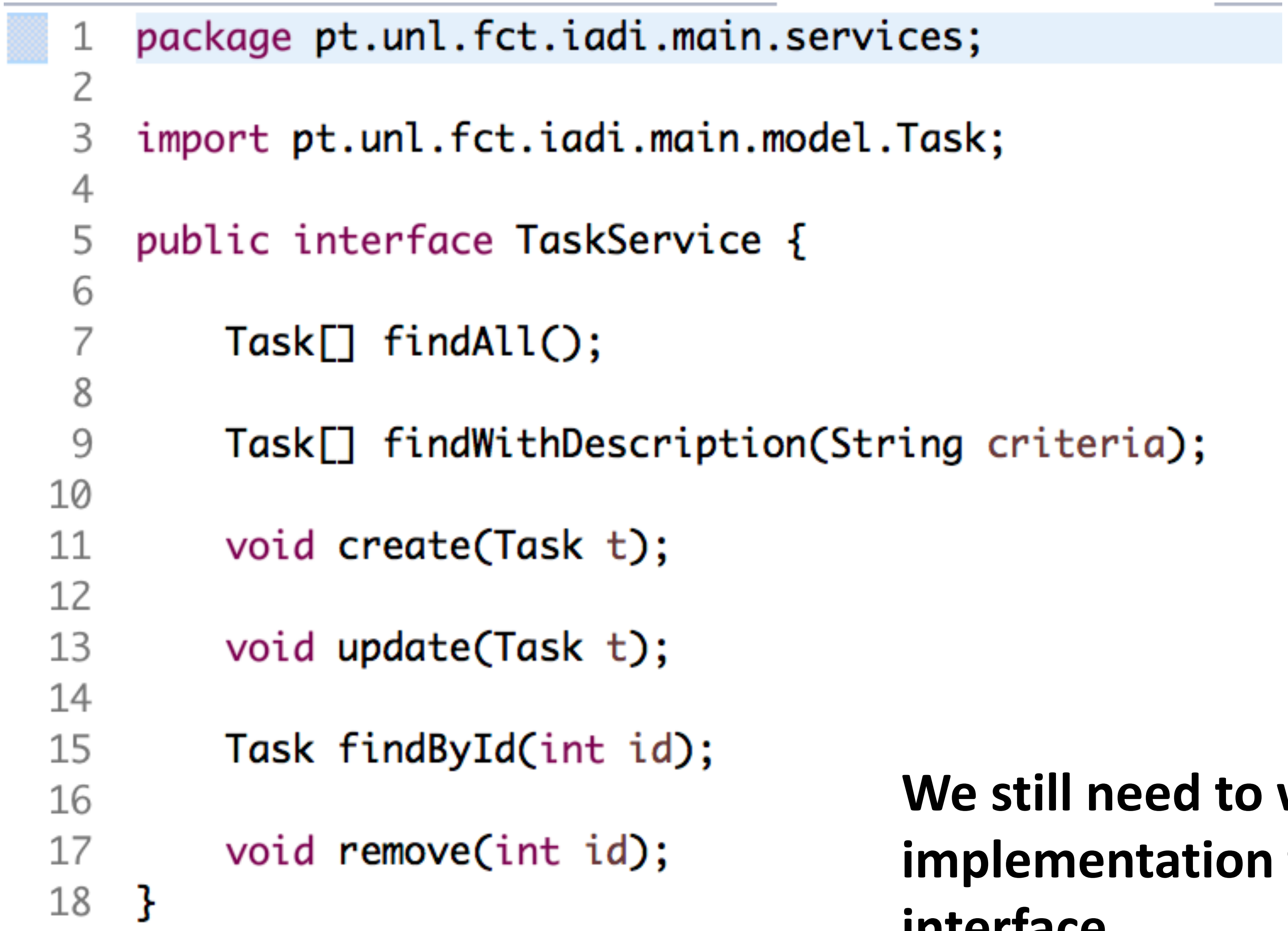
Spring by example…



```java
package pt.unl.fct.iadi.main.services;

import pt.unl.fct.iadi.main.model.Task;

public interface TaskService {

    Task[] findAll();

    Task[] findWithDescription(String criteria);

    void create(Task t);

    void update(Task t);

    Task findById(int id);

    void remove(int id);
}
```

Methods to Create Update and Delete a Task from the data storage system

Spring by example…

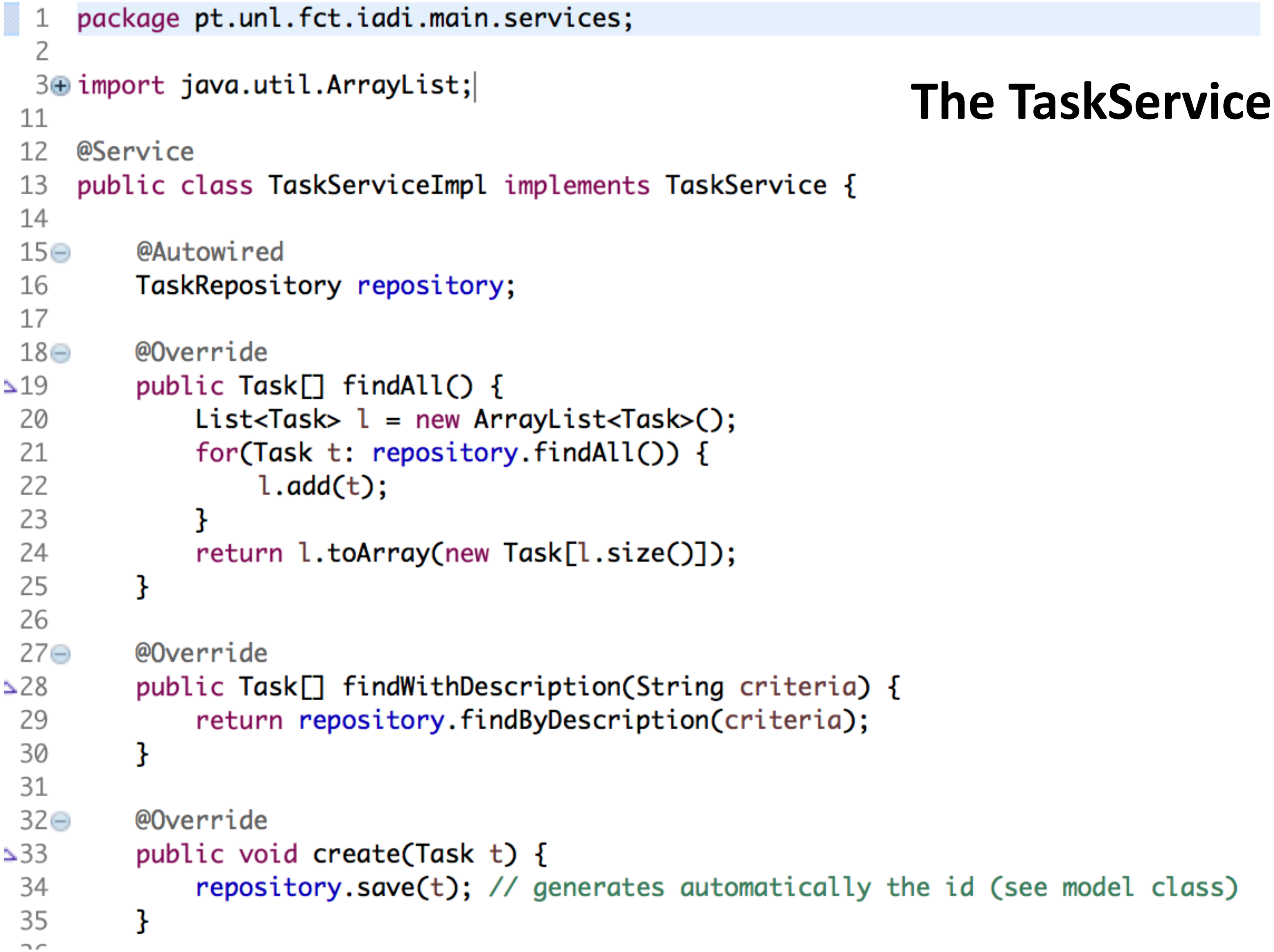


```java
package pt.unl.fct.iadi.main.services;

import pt.unl.fct.iadi.main.model.Task;

public interface TaskService {

    Task[] findAll();

    Task[] findWithDescription(String criteria);

    void create(Task t);

    void update(Task t);

    Task findById(int id);

    void remove(int id);
}
```

We still need to write the implementation for this interface.

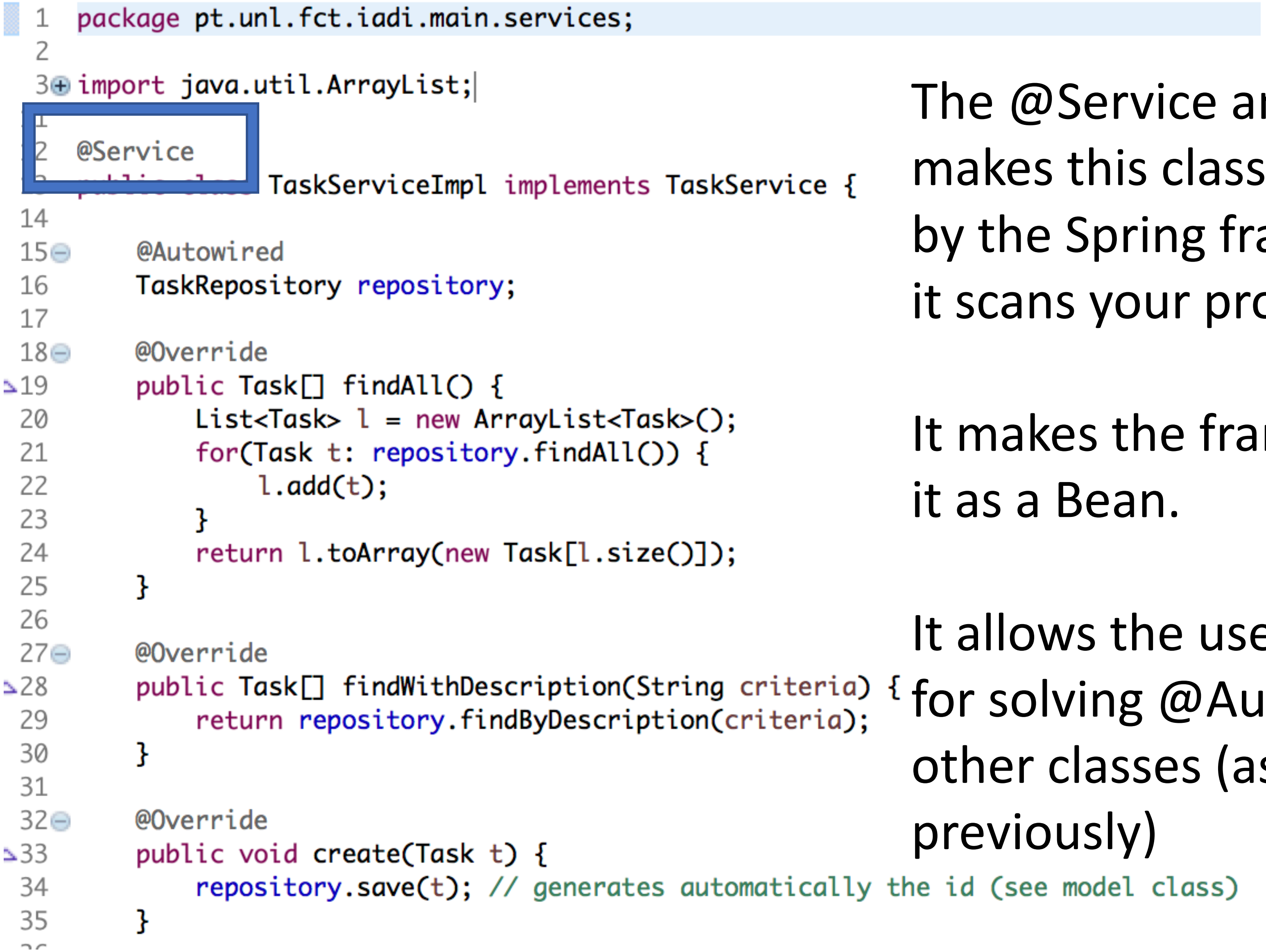
Spring by example…

The TaskServiceImpl Class



```java
package pt.unl.fct.iadi.main.services;

import java.util.ArrayList;

@Service
public class TaskServiceImpl implements TaskService {

    @Autowired
    TaskRepository repository;

    @Override
    public Task[] findAll() {
        List<Task> l = new ArrayList<Task>();
        for(Task t: repository.findAll()) {
            l.add(t);
        }
        return l.toArray(new Task[l.size()]);
    }

    @Override
    public Task[] findWithDescription(String criteria) {
        return repository.findByDescription(criteria);
    }

    @Override
    public void create(Task t) {
        repository.save(t); // generates automatically the id (see model class)
    }
```

Spring by example…



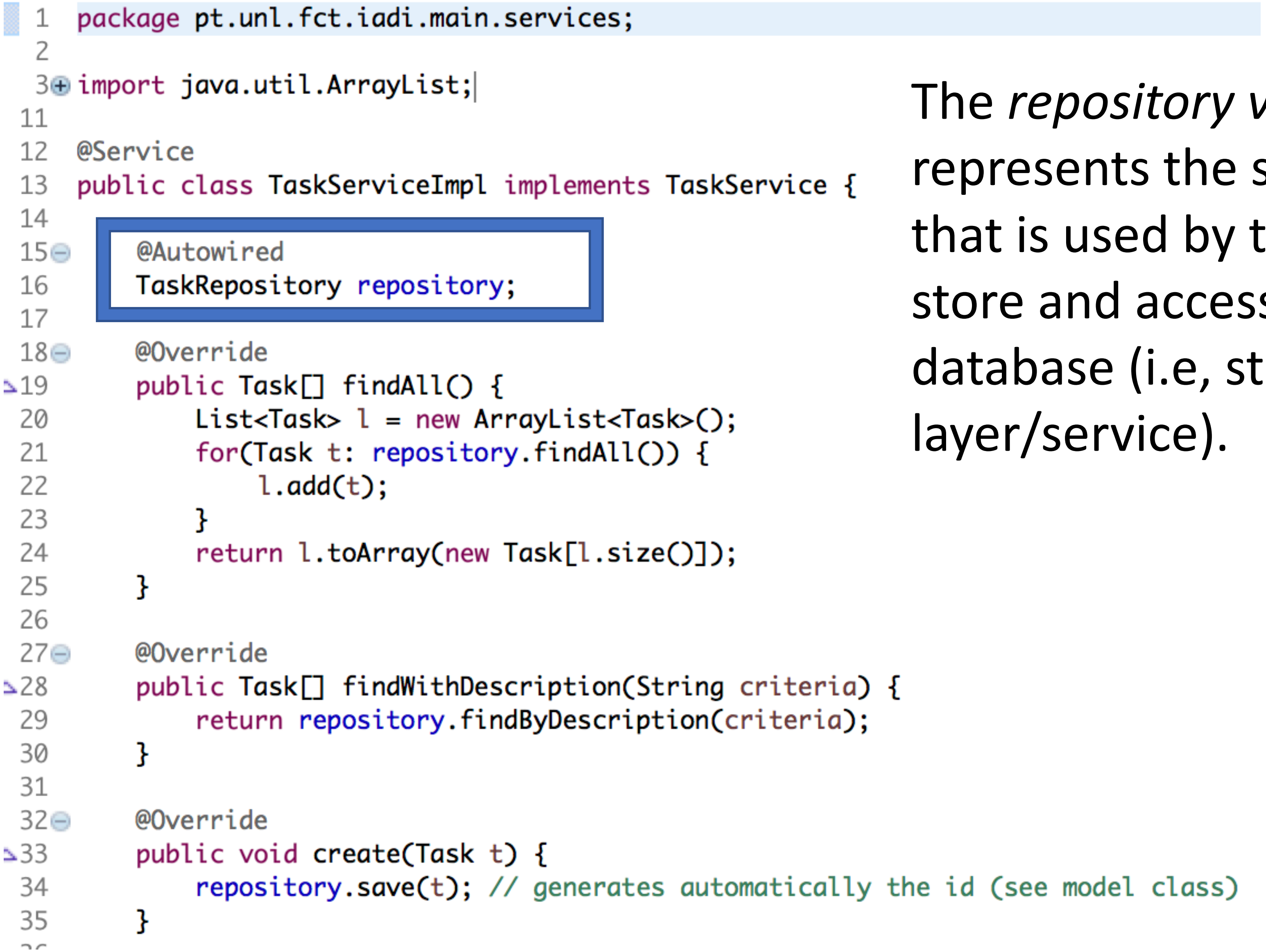
```java
package pt.unl.fct.iadi.main.services;

import java.util.ArrayList;

@Service
public class TaskServiceImpl implements TaskService {

    @Autowired
    TaskRepository repository;

    @Override
    public Task[] findAll() {
        List<Task> l = new ArrayList<Task>();
        for(Task t: repository.findAll()) {
            l.add(t);
        }
        return l.toArray(new Task[l.size()]);
    }

    @Override
    public Task[] findWithDescription(String criteria) {
        return repository.findByDescription(criteria);
    }

    @Override
    public void create(Task t) {
        repository.save(t); // generates automatically the id (see model class)
    }
```

The @Service annotation makes this class be inspected by the Spring framework when it scans your project.

It makes the framework export it as a Bean.

It allows the use of this class for solving @Autowired on other classes (as seen previously)

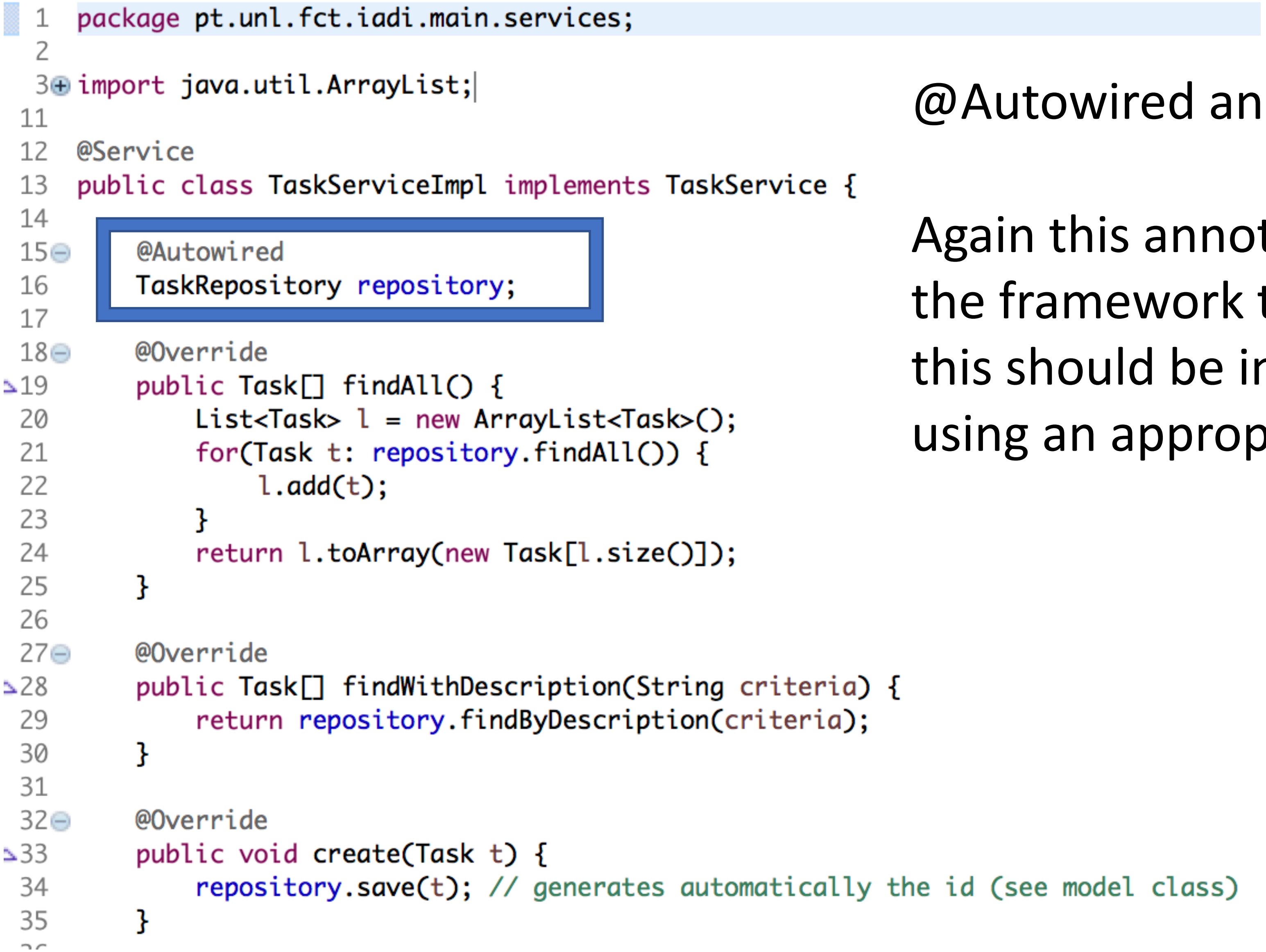
Spring by example…



```java
package pt.unl.fct.iadi.main.services;

import java.util.ArrayList;

@Service
public class TaskServiceImpl implements TaskService {

    @Autowired
    TaskRepository repository;

    @Override
    public Task[] findAll() {
        List<Task> l = new ArrayList<Task>();
        for(Task t: repository.findAll()) {
            l.add(t);
        }
        return l.toArray(new Task[l.size()]);
    }

    @Override
    public Task[] findWithDescription(String criteria) {
        return repository.findByDescription(criteria);
    }

    @Override
    public void create(Task t) {
        repository.save(t); // generates automatically the id (see model class)
    }
```

The *repository variable* represents the storage layer that is used by this service to store and access data in the database (i.e, storage layer/service).

Spring by example…

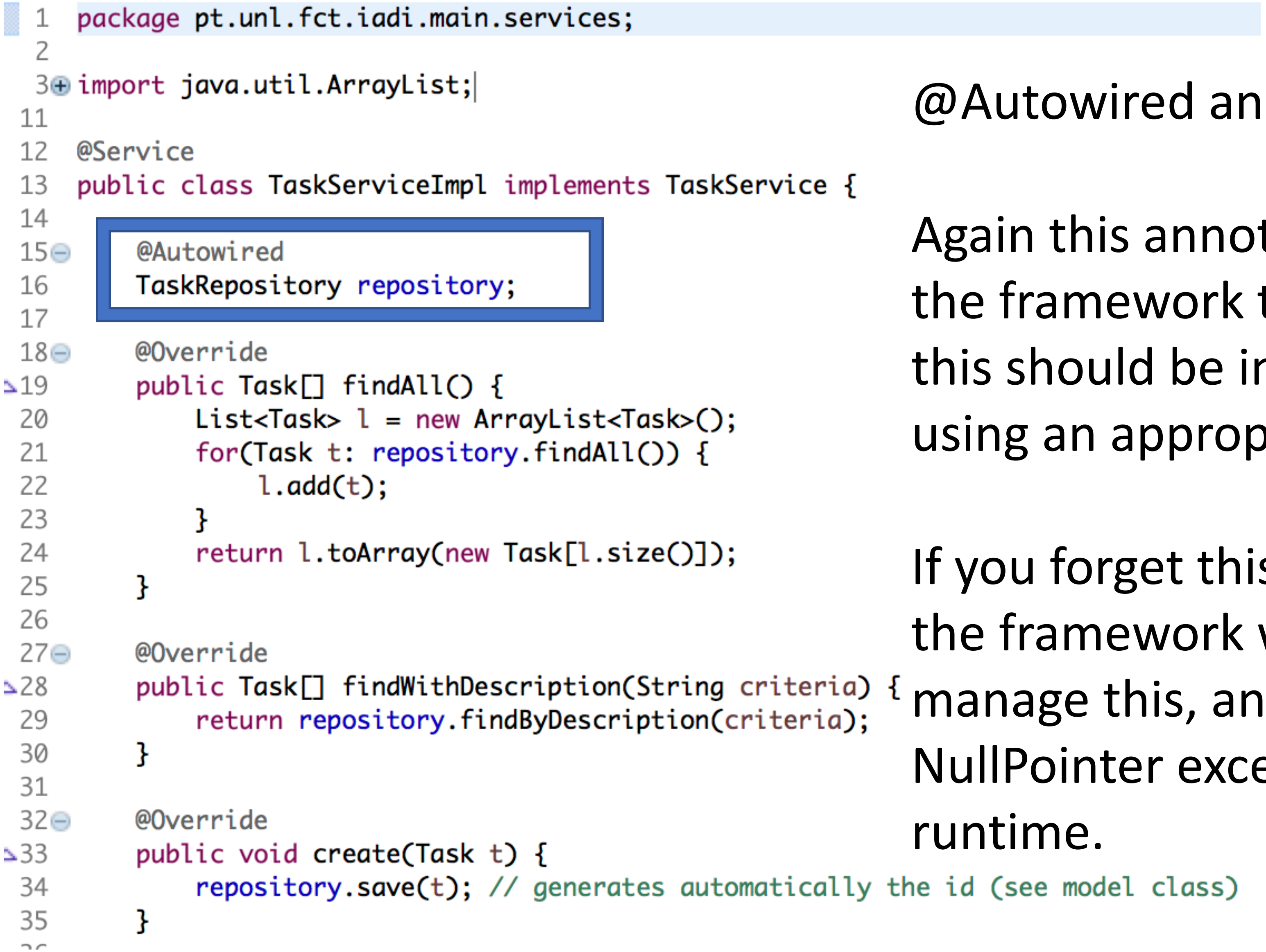


```java
package pt.unl.fct.iadi.main.services;

import java.util.ArrayList;

@Service
public class TaskServiceImpl implements TaskService {

    @Autowired
    TaskRepository repository;

    @Override
    public Task[] findAll() {
        List<Task> l = new ArrayList<Task>();
        for(Task t: repository.findAll()) {
            l.add(t);
        }
        return l.toArray(new Task[l.size()]);
    }

    @Override
    public Task[] findWithDescription(String criteria) {
        return repository.findByDescription(criteria);
    }

    @Override
    public void create(Task t) {
        repository.save(t); // generates automatically the id (see model class)
    }
```

@Autowired annotation

Again this annotation informs the framework that at runtime this should be instantiated using an appropriate instance.

Spring by example…



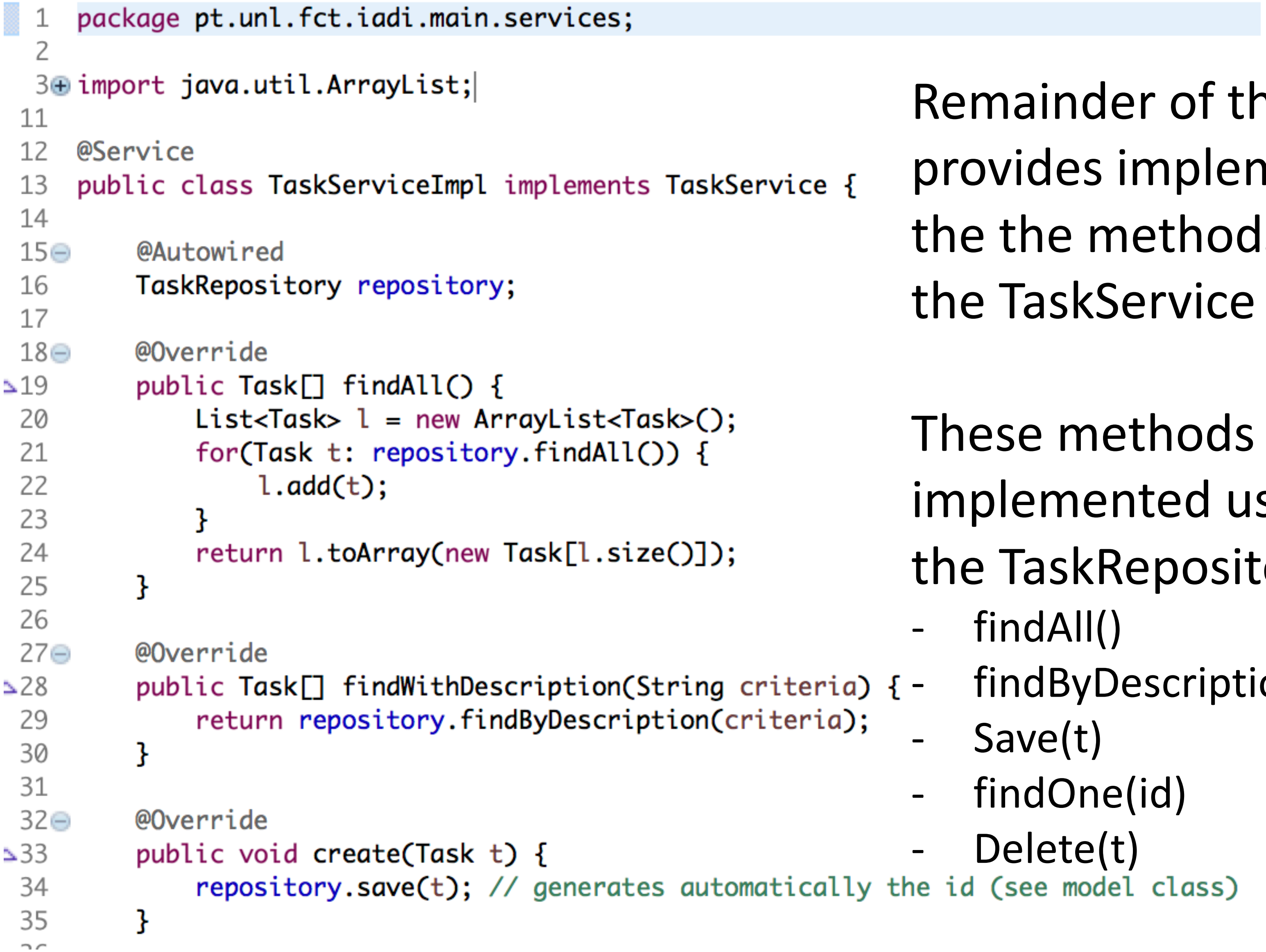
```java
package pt.unl.fct.iadi.main.services;

import java.util.ArrayList;

@Service
public class TaskServiceImpl implements TaskService {

    @Autowired
    TaskRepository repository;

    @Override
    public Task[] findAll() {
        List<Task> l = new ArrayList<Task>();
        for(Task t: repository.findAll()) {
            l.add(t);
        }
        return l.toArray(new Task[l.size()]);
    }

    @Override
    public Task[] findWithDescription(String criteria) {
        return repository.findByDescription(criteria);
    }

    @Override
    public void create(Task t) {
        repository.save(t); // generates automatically the id (see model class)
    }
```

@Autowired annotation

Again this annotation informs the framework that at runtime this should be instantiated using an appropriate instance.

If you forget this annotation the framework will not manage this, and you will get a NullPointer exception at runtime.

Spring by example…



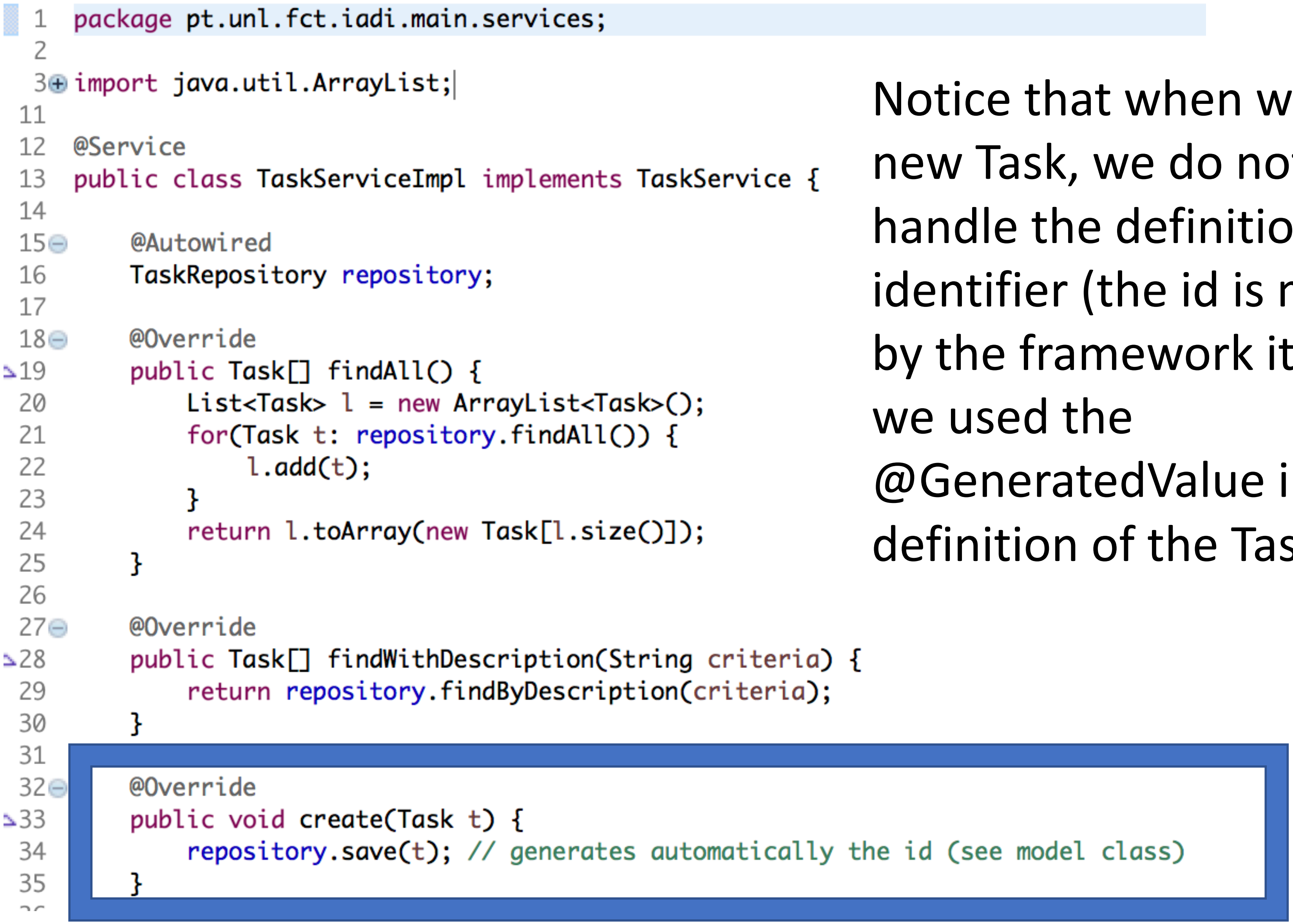
```
package pt.unl.fct.iadi.main.services;

import java.util.ArrayList;

@Service
public class TaskServiceImpl implements TaskService {

    @Autowired
    TaskRepository repository;

    @Override
    public Task[] findAll() {
        List<Task> l = new ArrayList<Task>();
        for(Task t: repository.findAll()) {
            l.add(t);
        }
        return l.toArray(new Task[l.size()]);
    }

    @Override
    public Task[] findWithDescription(String criteria) {
        return repository.findByDescription(criteria);
    }

    @Override
    public void create(Task t) {
        repository.save(t); // generates automatically the id (see model class)
    }
```

Remainder of the class provides implementations for the the methods defined in the TaskService interface.

These methods are implemented using method of the TaskRepository Interface:

- findAll()
- findByDescription(criteria)
- Save(t)
- findOne(id)
- Delete(t)

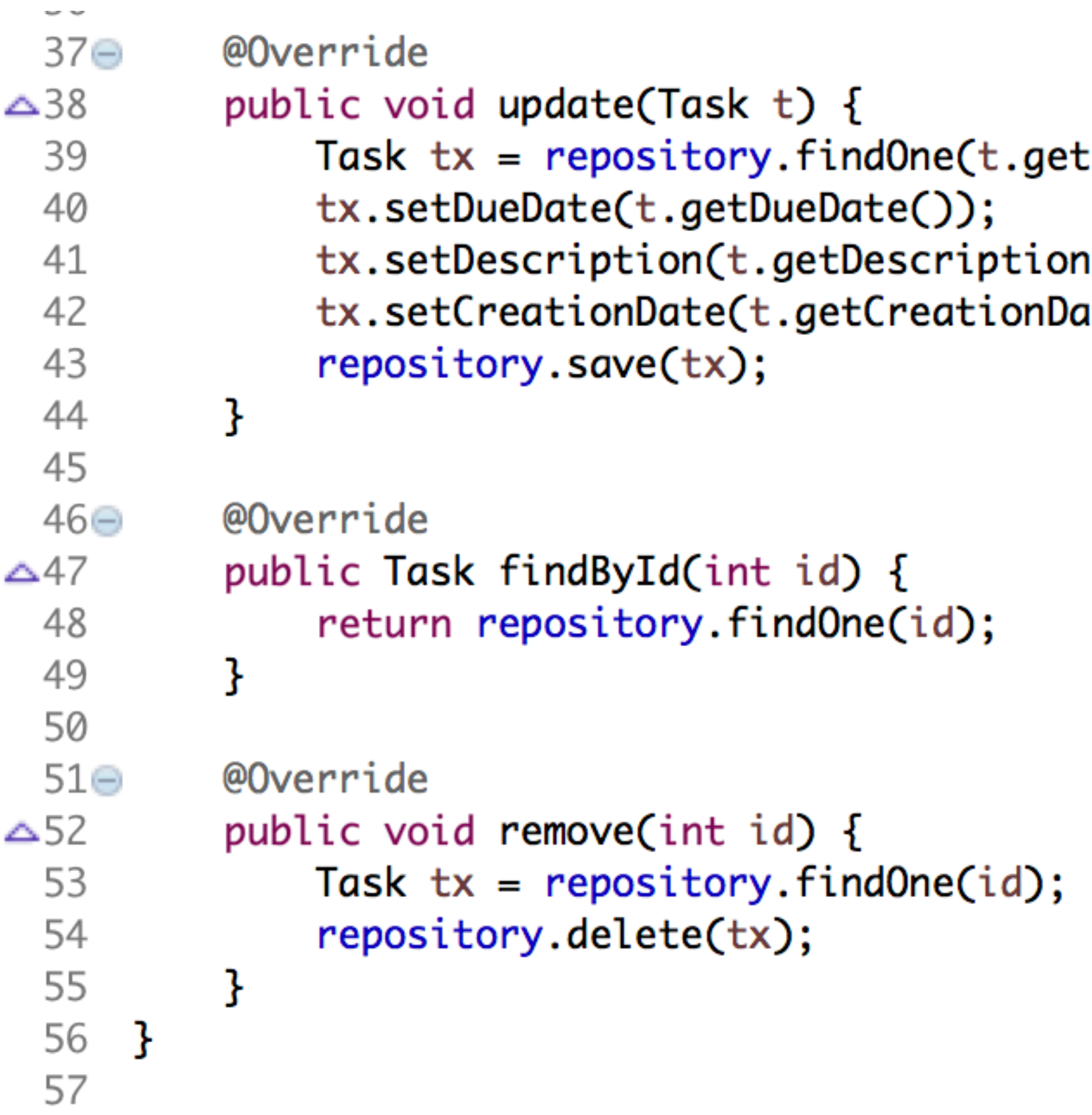
Spring by example…



```
package pt.unl.fct.iadi.main.services;

import java.util.ArrayList;

@Service
public class TaskServiceImpl implements TaskService {

    @Autowired
    TaskRepository repository;

    @Override
    public Task[] findAll() {
        List<Task> l = new ArrayList<Task>();
        for(Task t: repository.findAll()) {
            l.add(t);
        }
        return l.toArray(new Task[l.size()]);
    }

    @Override
    public Task[] findWithDescription(String criteria) {
        return repository.findByDescription(criteria);
    }

    @Override
    public void create(Task t) {
        repository.save(t); // generates automatically the id (see model class)
    }
```

Notice that when we store a new Task, we do not need to handle the definition of the identifier (the id is managed by the framework itself, since we used the @GeneratedValue in the definition of the Task class.

Spring by example…



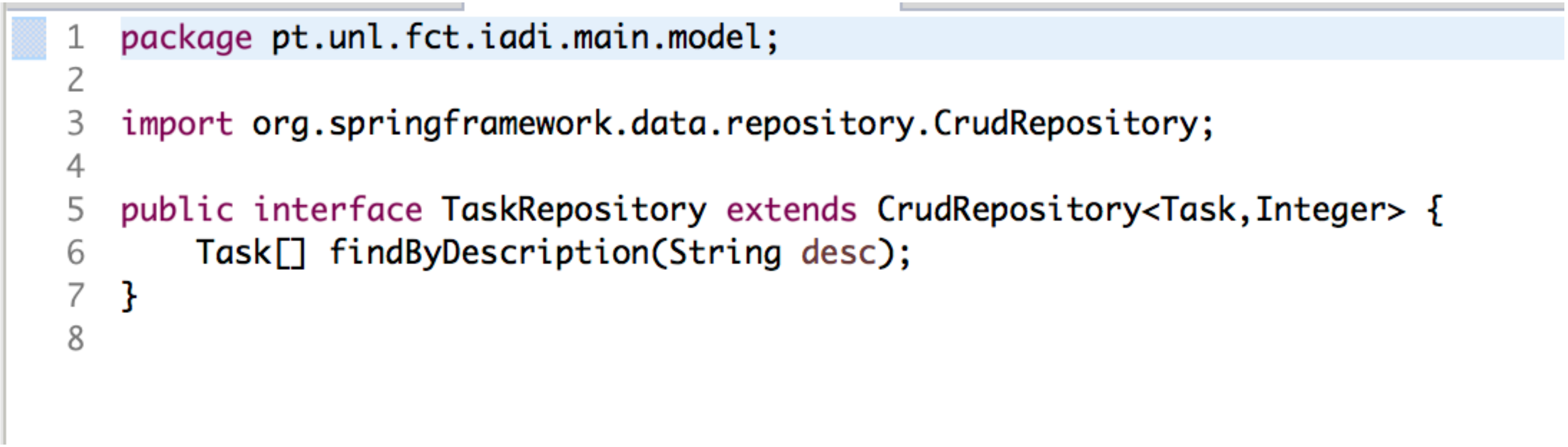
```
    @Override
    public void update(Task t) {
        Task tx = repository.findOne(t.getId());
        tx.setDueDate(t.getDueDate());
        tx.setDescription(t.getDescription());
        tx.setCreationDate(t.getCreationDate());
        repository.save(tx);
    }

    @Override
    public Task findById(int id) {
        return repository.findOne(id);
    }

    @Override
    public void remove(int id) {
        Task tx = repository.findOne(id);
        repository.delete(tx);
    }
}
```

Remainder of the class provides implementations for the the methods defined in the TaskService interface.

These methods are implemented using method of the TaskRepository Interface:

- findAll()
- findByDescription(criteria)
- Save(t)
- findOne(id)
- Delete(t)

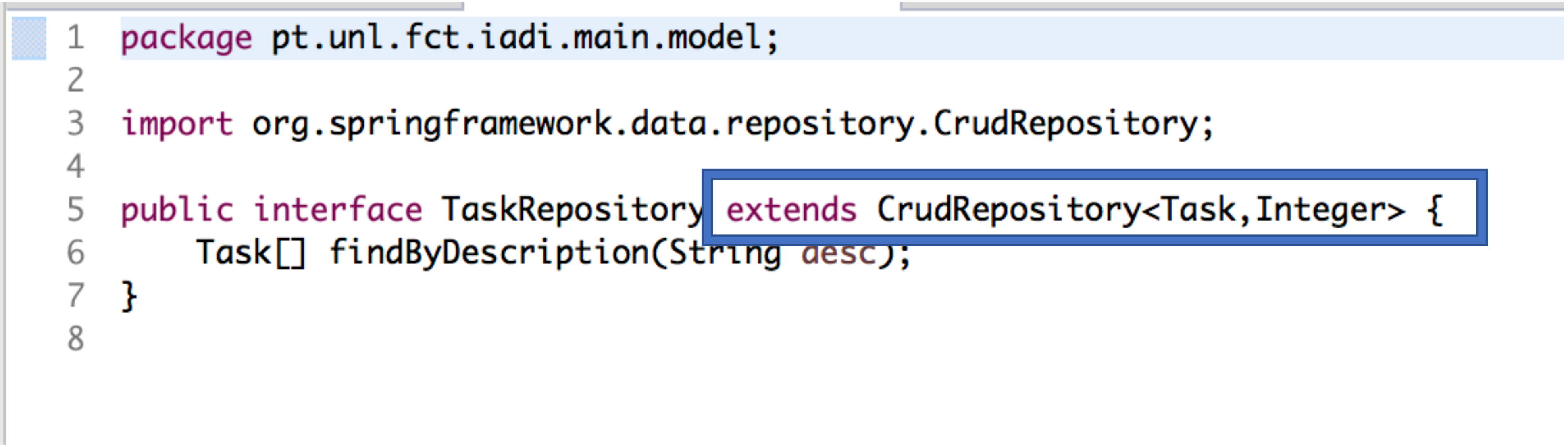
Spring by example…



```java
package pt.unl.fct.iadi.main.model;

import org.springframework.data.repository.CrudRepository;

public interface TaskRepository extends CrudRepository<Task,Integer> {
    Task[] findByDescription(String desc);
}

```

The TaskRepository Interface

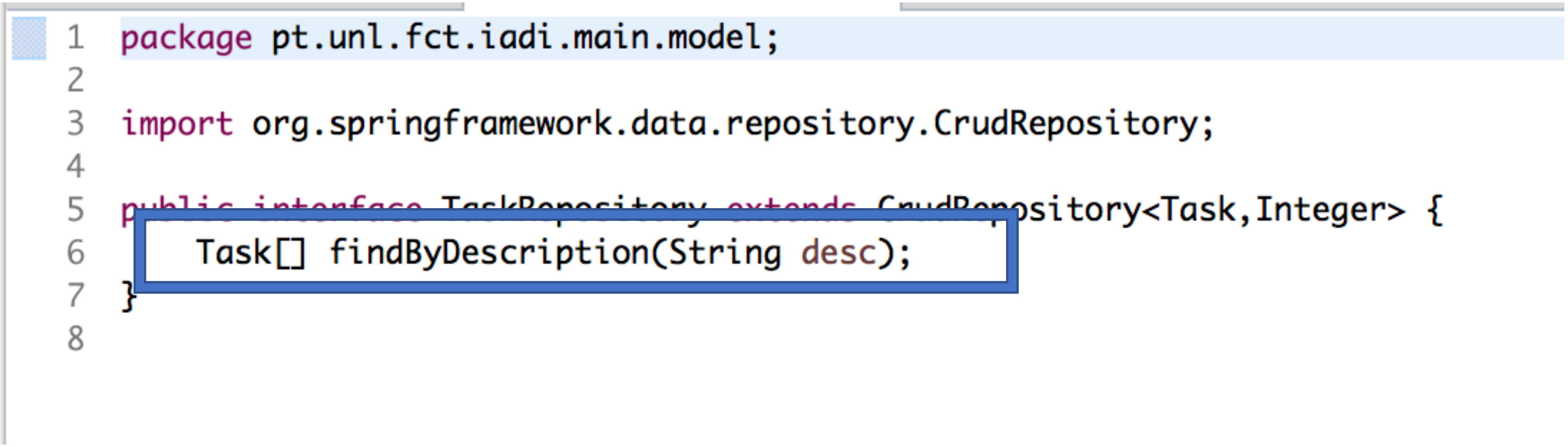
Spring by example…



```
package pt.unl.fct.iadi.main.model;

import org.springframework.data.repository.CrudRepository;

public interface TaskRepository extends CrudRepository<Task,Integer> {
    Task[] findByDescription(String desc);
}

```

It extends another Interface named CrudRepository that is typified with Task and Integer. This indicates that this interface is the interface for a Repository (data storage system, usually identified by the @Repository annotation) that is specialized to store instances of the Task entity, identified by Integers (Primary Key).

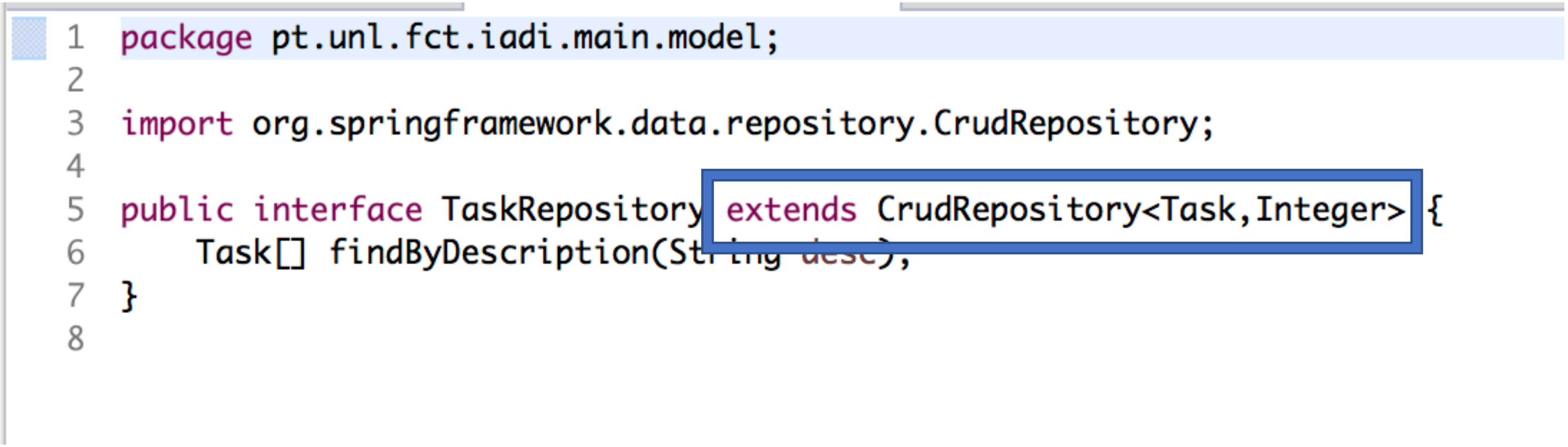
Spring by example…



```java
package pt.unl.fct.iadi.main.model;

import org.springframework.data.repository.CrudRepository;

public interface TaskRepository extends CrudRepository<Task,Integer> {
    Task[] findByDescription(String desc);
}

```

It extends another Interface named CrudRepository that is typified with Task and Integer. This indicates that this interface is the interface for a Repository (data storage system, usually identified by the @Repository annotation) that is specialized to store instances of the Task entity, identified by Integers (Primary Key).

Spring by example…

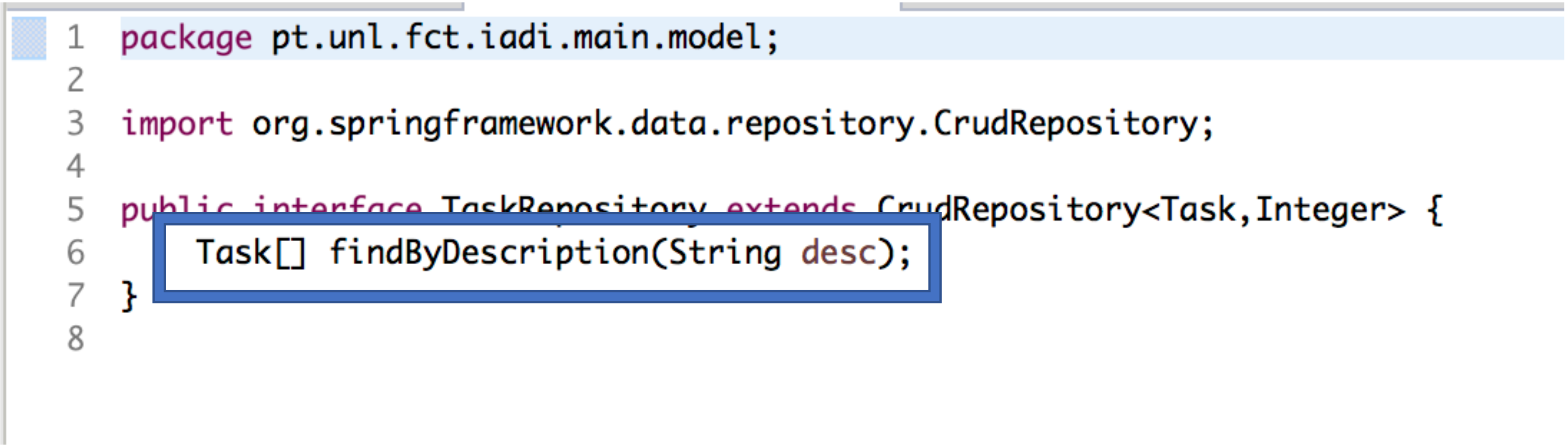


```java
package pt.unl.fct.iadi.main.model;

import org.springframework.data.repository.CrudRepository;

public interface TaskRepository extends CrudRepository<Task,Integer> {
    Task[] findByDescription(String desc);
}

```

CRUD Interface implies that there are methods to:

Create
Read
Update
Delete

Spring by example…

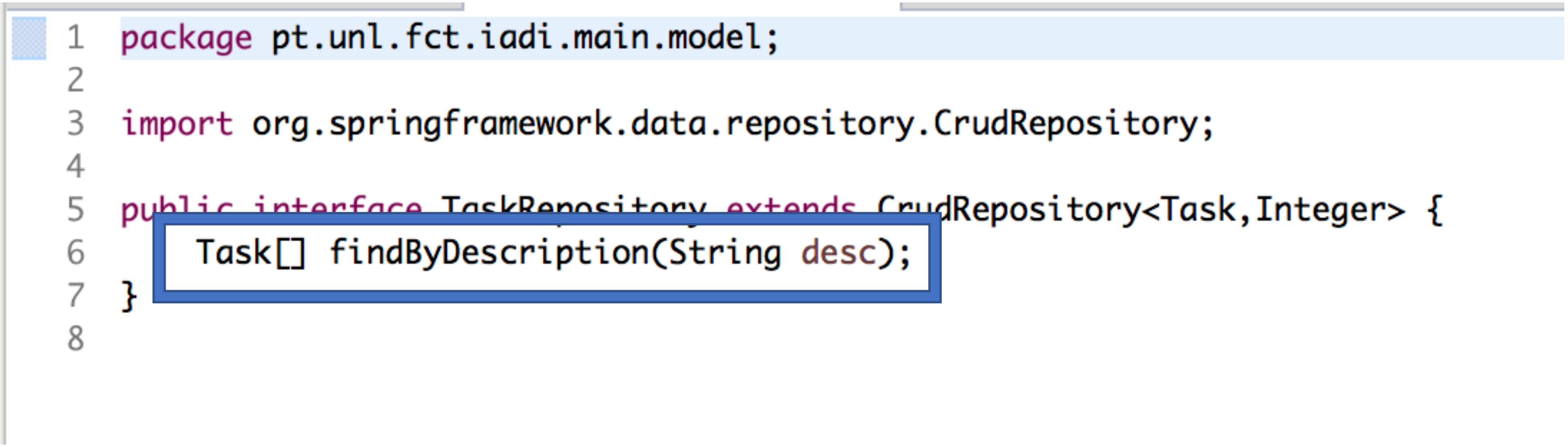


```java
package pt.unl.fct.iadi.main.model;

import org.springframework.data.repository.CrudRepository;

public interface TaskRepository extends CrudRepository<Task,Integer> {
    Task[] findByDescription(String desc);
}

```

We are specifying that our interface will have all methods of the CrudRepository Interface, and another one called:

findByDescription(String desc) that returns all Tasks with the value desc in the Description field.

Spring by example…



```java
package pt.unl.fct.iadi.main.model;

import org.springframework.data.repository.CrudRepository;

public interface TaskRepository extends CrudRepository<Task,Integer> {
    Task[] findByDescription(String desc);
}

```

There are no annotations but the framework here uses the name of the method to infer what code should be automatically generated to support it.

In particular this entails generating a SQL query to query the database with an equality filter. Seems magic… but its not...

Spring by example…

- So now you can run the Application class (if you check the code of the Application class it now creates a few tasks when it boots to allow you to see some state in the database). And everything will be working...

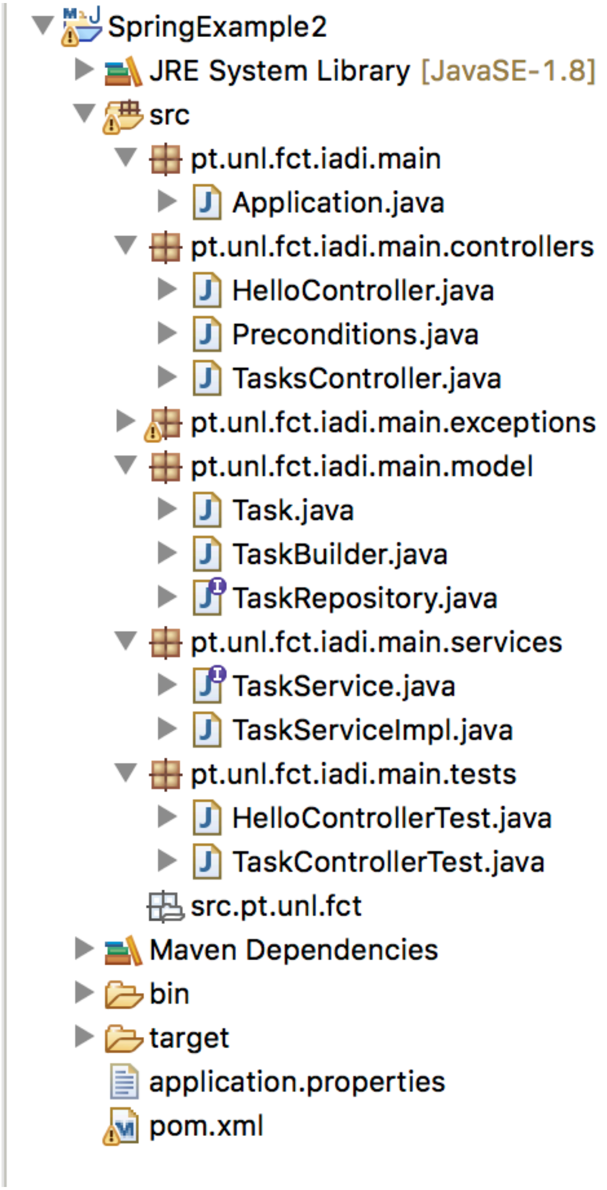
- You can use the REST interface to manipulate tasks, and these are stored in a local instance of the H2 database that the framework started (You can check the output of the Application to make sure of this).

Spring by example…

- Wait…

- How did the framework did know that it should use H2?

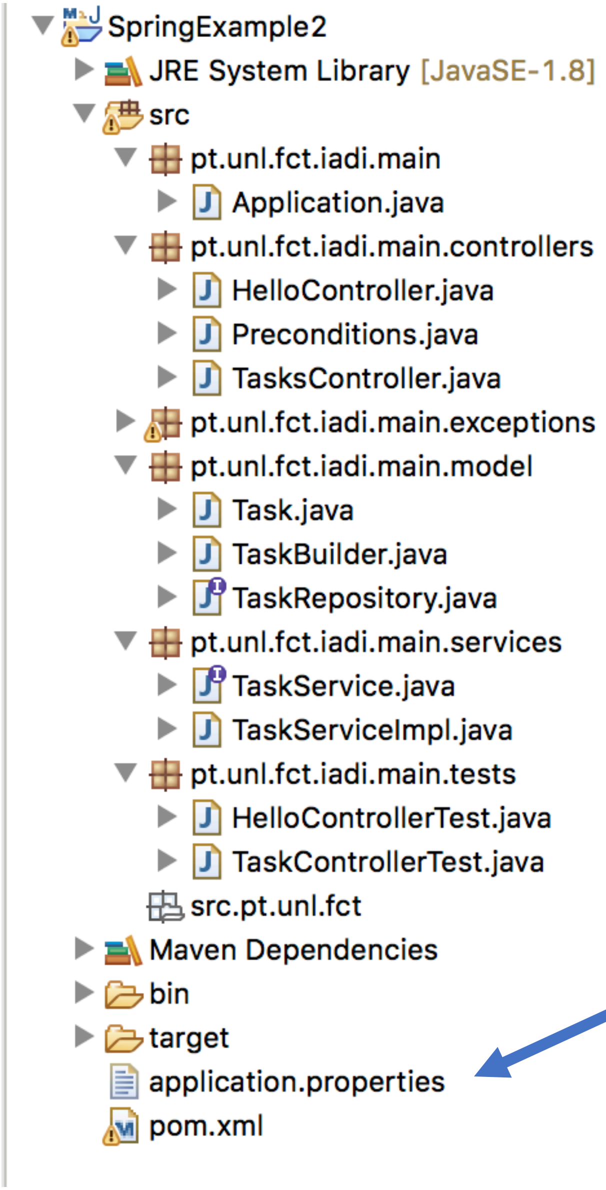
Spring by example...



```
▼ SpringExample2
  ▶ JRE System Library [JavaSE-1.8]
  ▼ src
    ▼ pt.unl.fct.iadi.main
      ▶ Application.java
    ▼ pt.unl.fct.iadi.main.controllers
      ▶ HelloController.java
      ▶ Preconditions.java
      ▶ TasksController.java
    ▶ pt.unl.fct.iadi.main.exceptions
    ▼ pt.unl.fct.iadi.main.model
      ▶ Task.java
      ▶ TaskBuilder.java
      ▶ TaskRepository.java
    ▼ pt.unl.fct.iadi.main.services
      ▶ TaskService.java
      ▶ TaskServiceImpl.java
    ▼ pt.unl.fct.iadi.main.tests
      ▶ HelloControllerTest.java
      ▶ TaskControllerTest.java
      src.pt.unl.fct
  ▶ Maven Dependencies
  ▶ bin
  ▶ target
    application.properties
    pom.xml
```

The answer to this lies in a few files that we did not inspect yet...

Spring by example…



```
SpringExample2
  JRE System Library [JavaSE-1.8]
  src
    pt.unl.fct.iadi.main
      Application.java
    pt.unl.fct.iadi.main.controllers
      HelloController.java
      Preconditions.java
      TasksController.java
    pt.unl.fct.iadi.main.exceptions
    pt.unl.fct.iadi.main.model
      Task.java
      TaskBuilder.java
      TaskRepository.java
    pt.unl.fct.iadi.main.services
      TaskService.java
      TaskServiceImpl.java
    pt.unl.fct.iadi.main.tests
      HelloControllerTest.java
      TaskControllerTest.java
    src.pt.unl.fct
  Maven Dependencies
  bin
  target
  application.properties
  pom.xml
```

This files specifies properties used at runtime by your application.

Spring by example…

application.properties

```
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE
```

Spring by example...

application.properties

```
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE
```

Actually this is just stating that interactions with the database through SQL should be logged...

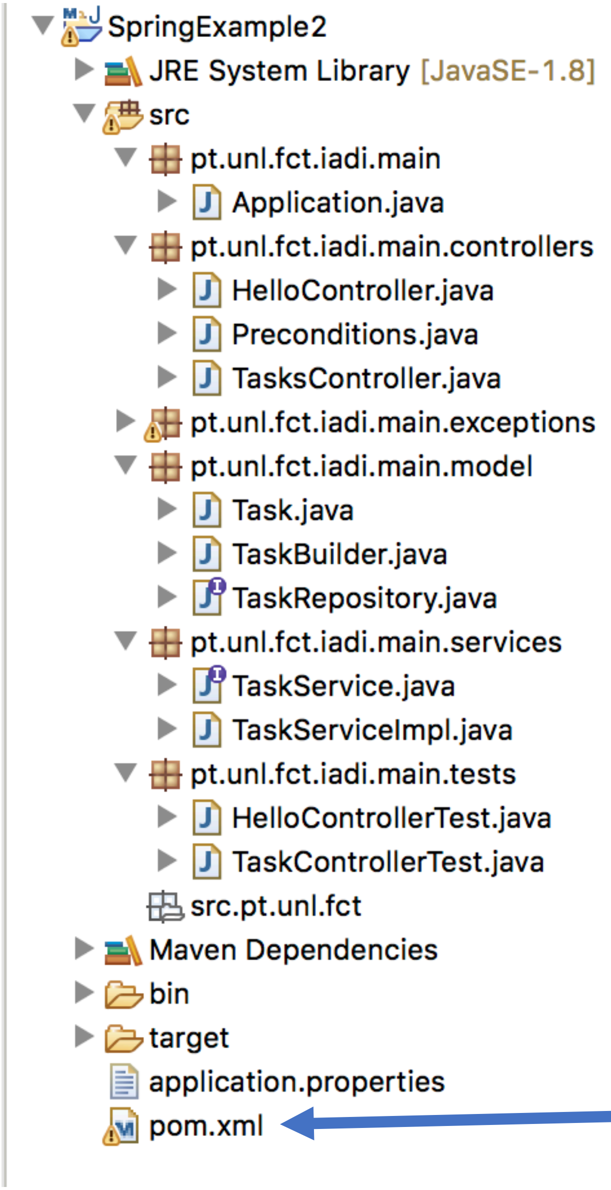
Spring by example...

application.properties

```
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE
```

But if your database was running on some other machiene and protected by credentials you could define that here...

Spring by example…

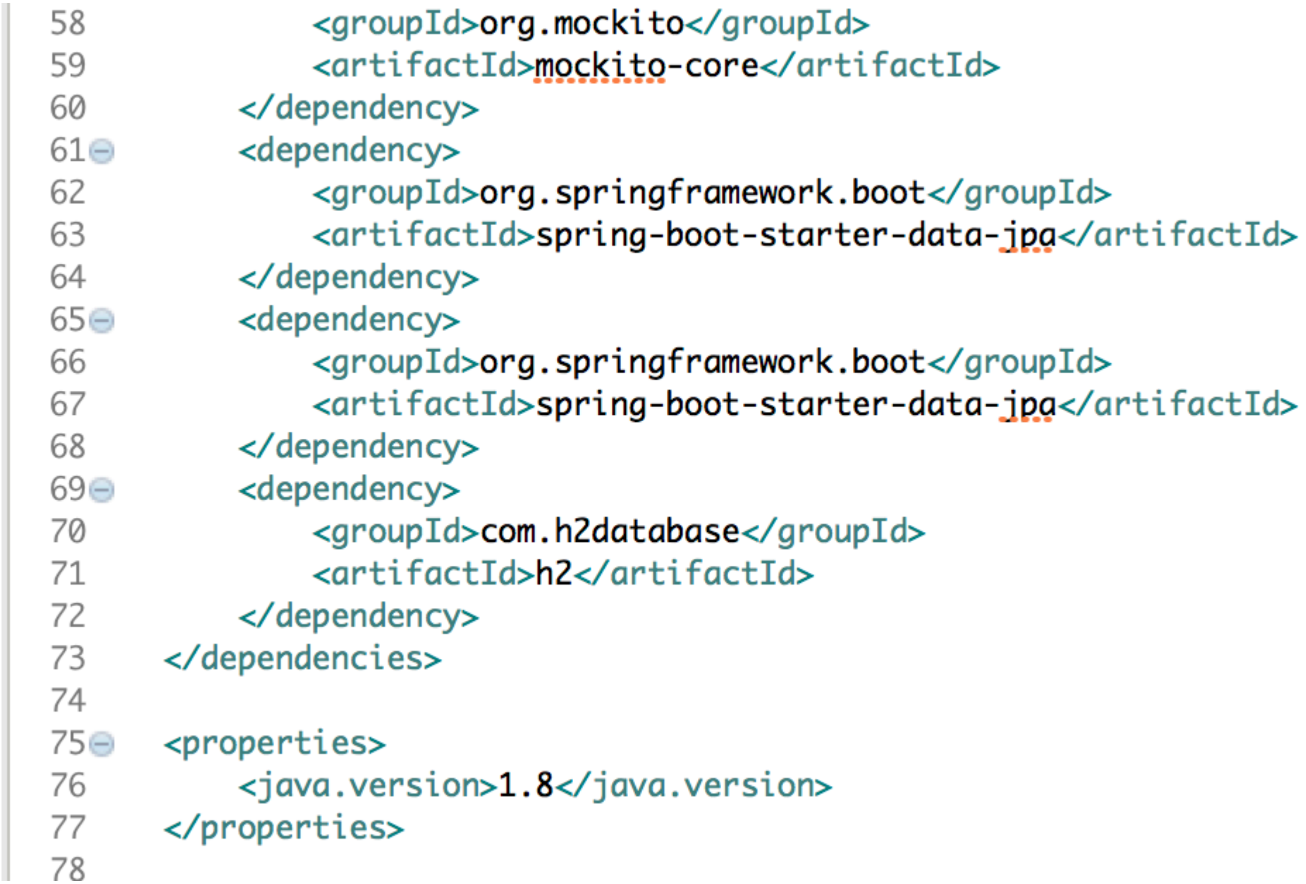


```
▼ SpringExample2
  ▶ JRE System Library [JavaSE-1.8]
  ▼ src
    ▼ pt.unl.fct.iadi.main
      ▶ Application.java
    ▼ pt.unl.fct.iadi.main.controllers
      ▶ HelloController.java
      ▶ Preconditions.java
      ▶ TasksController.java
    ▶ pt.unl.fct.iadi.main.exceptions
    ▼ pt.unl.fct.iadi.main.model
      ▶ Task.java
      ▶ TaskBuilder.java
      ▶ TaskRepository.java
    ▼ pt.unl.fct.iadi.main.services
      ▶ TaskService.java
      ▶ TaskServiceImpl.java
    ▼ pt.unl.fct.iadi.main.tests
      ▶ HelloControllerTest.java
      ▶ TaskControllerTest.java
    src.pt.unl.fct
  ▶ Maven Dependencies
  ▶ bin
  ▶ target
    application.properties
    pom.xml
```

pom.xml defines the dependencies of our application

Spring by example…

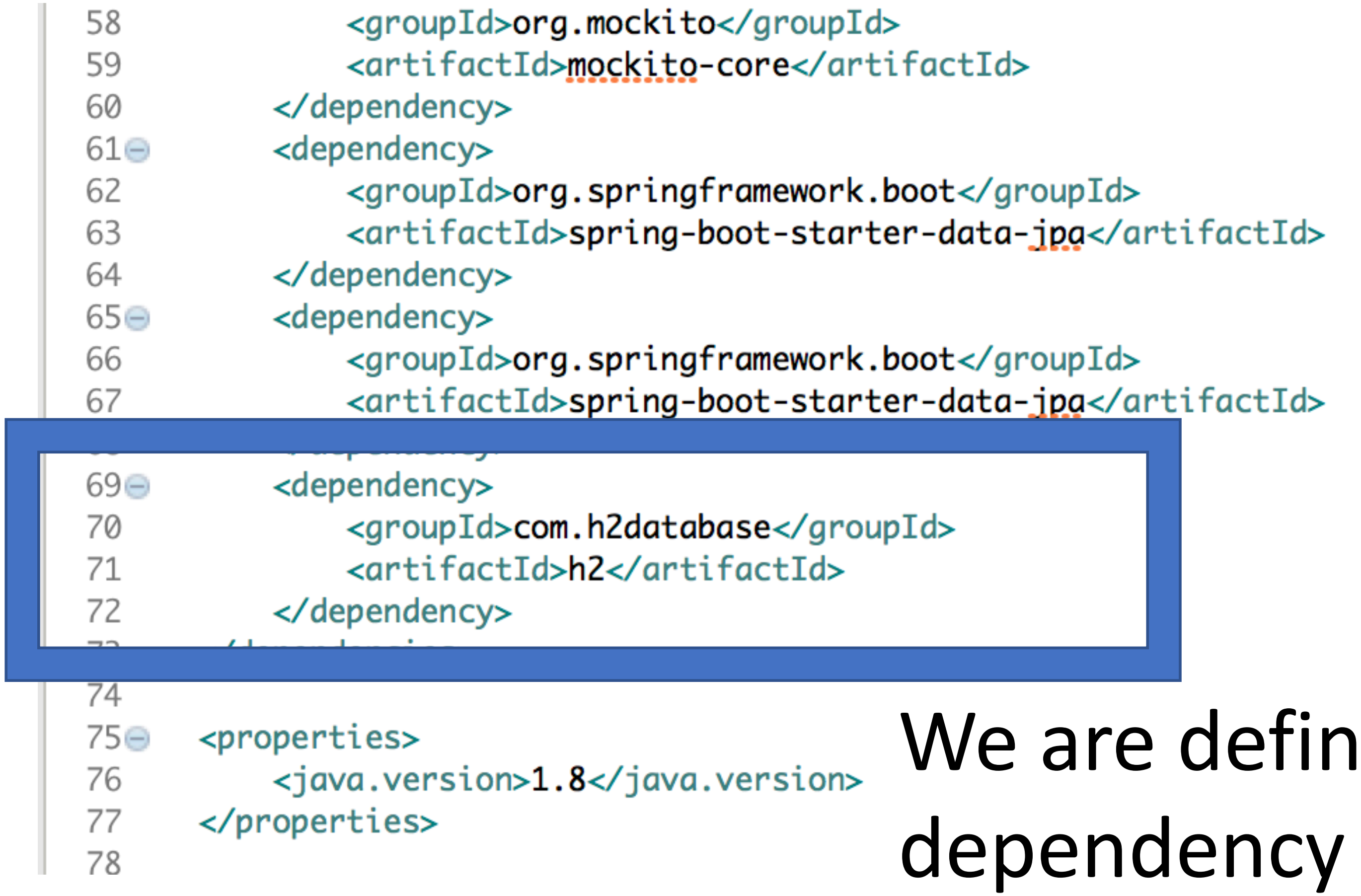
pom.xml (partial)



```
            <groupId>org.mockito</groupId>
            <artifactId>mockito-core</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
        </dependency>
    </dependencies>

    <properties>
        <java.version>1.8</java.version>
    </properties>

```

Spring by example…

pom.xml (partial)

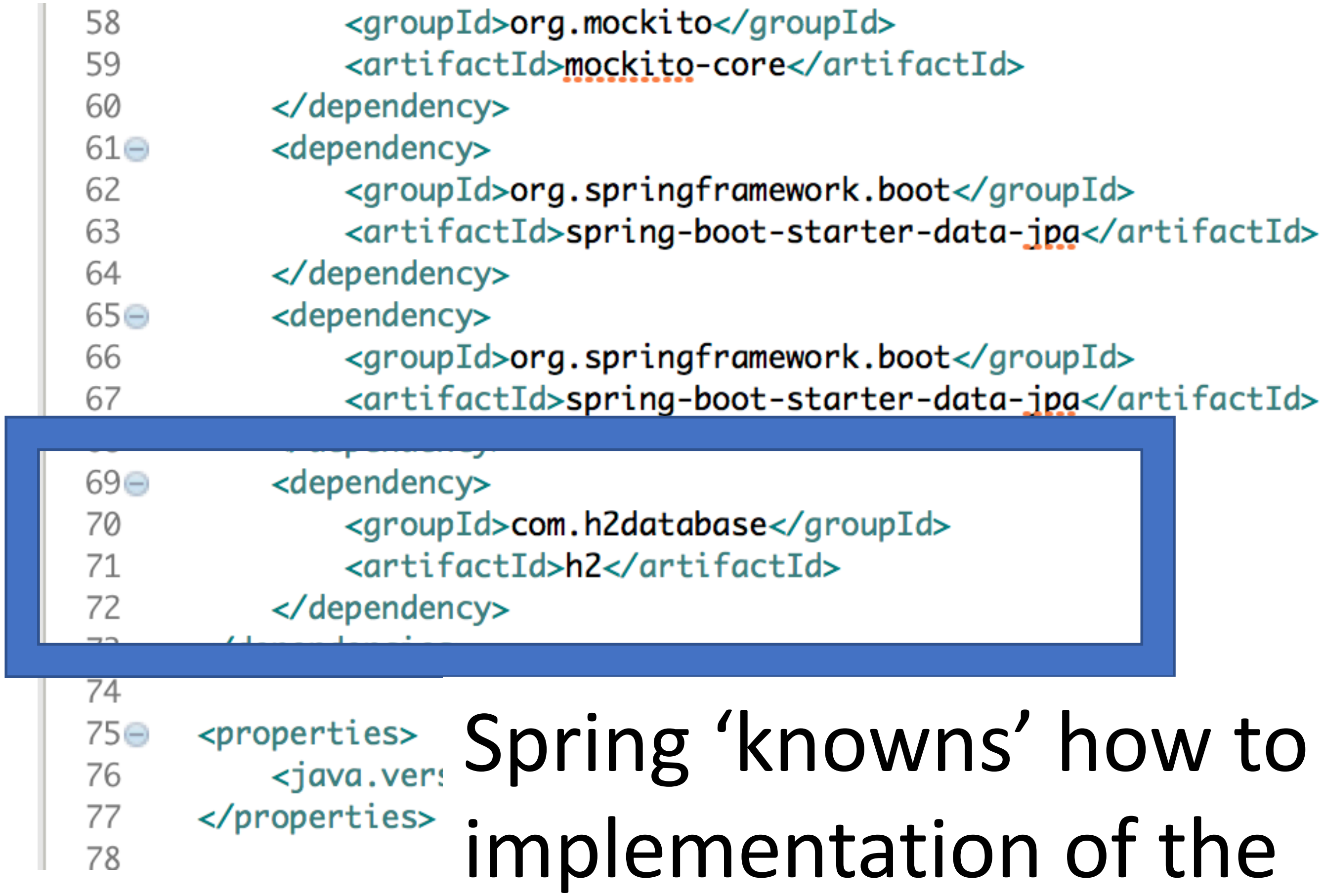


```
            <groupId>org.mockito</groupId>
            <artifactId>mockito-core</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
        </dependency>
    </dependencies>

    <properties>
        <java.version>1.8</java.version>
    </properties>

```

We are defining a dependency of the H2Database.

Spring by example...

pom.xml (partial)

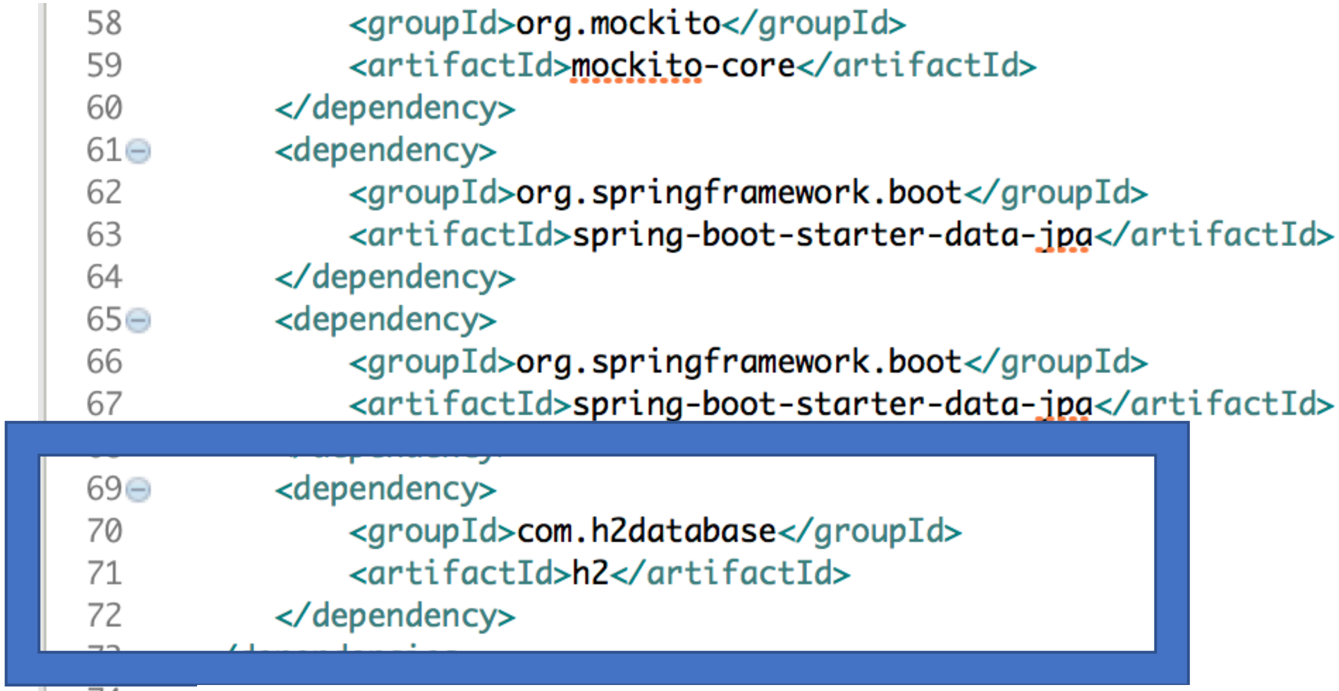


```
            <groupId>org.mockito</groupId>
            <artifactId>mockito-core</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
        </dependency>
    </dependencies>

    <properties>
        <java.ver
    </properties>
```

Spring ‘knowns’ how to instanciate an implementation of the CRUDRepository compatible with H2.

Spring by example...

pom.xml (partial)



```
            <groupId>org.mockito</groupId>
            <artifactId>mockito-core</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
        </dependency>
    </dependencies>
```

It also lauchs H2 for you, and since this is a SQL database, it derives the tables that you need to support your entities.

Spring by example...

- Ok... lets Test this...
- Put the Application running (if it not running yet) and lets exercise the endpoints presented before.
- Use Postman to do these tests...
- Play with it for a while...

Reminder of the Class

- Project...

- Start working on materializing the entities manipulated by your application.
- Again maybe focus on ArtPieces for a start.
- Use H2 to develop and test your application (switching to MySQL later on will be relatively simple and something done in a self contained way).