codecademy

Cheatsheets / **Responding to Requests**

Spring Controllers

Mapping HTTP Requests

The `@RequestMapping` annotation can be used at the method level or the class level to map an HTTP request to the appropriate controller method.

```
@RequestMapping("/sayhello")
public String sayHello() {
  return "Hello, world";
}
```

Base Path Mapping

When the `@RequestMapping` annotation is used at the class level, the specified `path` attribute becomes the base path for the class.
In the example code, `getallRecipes` is called for every GET request to the `/foodierecipes` endpoint.

```
@RequestMapping("/foodierecipes")
public class FoodieRecipesController {

  private final RecipeRepository
recipeRepository;

  public
FoodieRecipesController(RecipeRepository
recipeRepo) {
    this.recipeRepository = recipeRepo;
  }

  @GetMapping()
  public Iterable<Recipe> getAllRecipes()
{
    return
this.recipeRepository.findAll();
  }
}
```

Common Request Types

Spring provides annotations that map to common request types. These methods include `@GetMapping`, `@PostMapping`, `@PutMapping`, and `@DeleteMapping`.

```
// Method parameters and bodies omitted
for brevity

@RestController
public class FlowerController {

  @GetMapping("/flowers")
  public Iterable<Flower> getAllFlowers()
{}

  @PostMapping("/flowers")
  public Flower addFlower() {}

  @PutMapping("/flowers/{id}")
  public Flower editFlower() {}

  @DeleteMapping("/flowers/{id}")
  public Flower deleteFlower() {}
}
```

Accessing Parameters in Methods

The `@RequestParam` annotation can be used at the method parameter level to allow the HTTP request parameters to be accessed in the method.

```
// Accepts GET requests to /fruit?
fruitType=mango

@GetMapping("/fruit")
public fruit
isFruitAvailable(@RequestParam String
fruitType) {
  return fruit.find(fruitType);
}
```

REST Controllers

`@RestController` is a class level annotation used to combine the functionality of the `@Controller` and `@ResponseBody` annotations.

- `@Controller` designates the annotated class as a controller
- `@ResponseBody` allows returned objects to be automatically serialized into JSON and returned in the HTTP response body

```
@RestController
public class LocationController {

  @GetMapping("/{gpsCoordinates}")
  public City getByCoordinates(@PathVariable String gpsCoordinates) {
    return this.locations.findByCoordinates(gpsCoordinates);
  }

}
```

Response Exceptions

Spring controllers can return a custom HTTP status code by throwing an instance of `ResponseStatusException`, which accepts an argument of type `HttpStatus`.

```
@GetMapping("/{id}")
public Book isBookAvailable(@PathVariable string id)
{
  if (id.isNumeric()) {
    int idAsInteger = Integer.parseInt(id)
    return book.findByID(idAsInteger)
  }
  else {
    throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "The ID contained a non-numerical value.");
  }
}
```

HttpStatus Type

In Spring, the `HttpStatus` type can be used to represent different HTTP status codes.

```
HttpStatus.OK // 200 code

HttpStatus.MOVED_PERMANENTLY // 301 code

HttpStatus.NOT_FOUND // 404 code

HttpStatus.BAD_GATEWAY // 502 code
```

Spring Specifying HTTP Status Code

In Spring, we have the option of apply the `@ResponseStatus` annotation to a method to designate a specific `HttpStatus`.

```
@PostMapping("/book")
@ResponseStatus(HttpStatus.CREATED)
public void addNewBook(@RequestParam
string title) {
  this.library.add(title);
}
```

Deserializing to an Object

In Spring, applying the `@RequestBody` annotation to a controller's method enables automatic deserialization of the HTTP request body to an object bound to the method's argument.

```
@GetMapping("/book")
public Book isBookAvailable(@RequestBody
Book book) {
  return library.find(book);
}
```

Print Share