

Criando o Primeiro Serviço



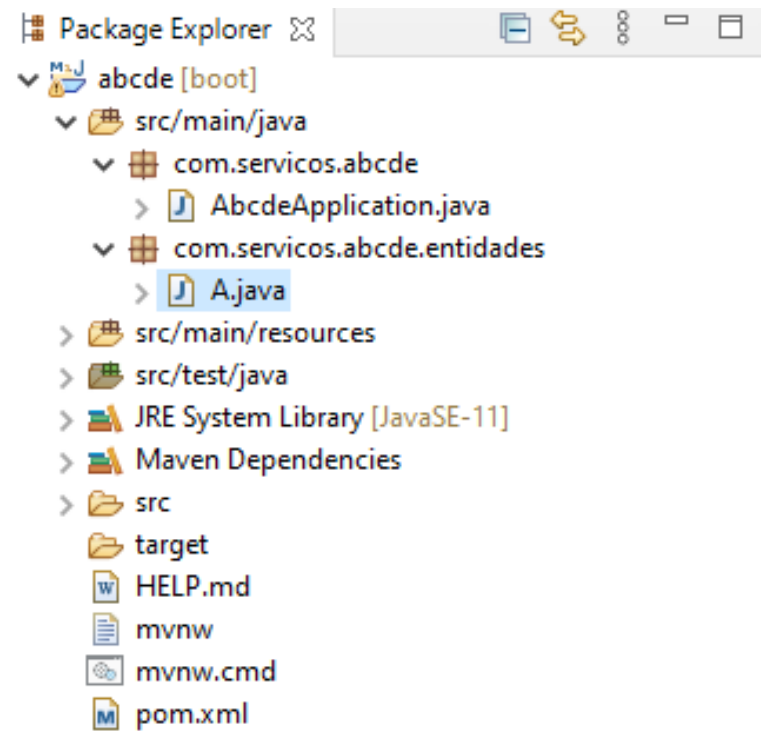
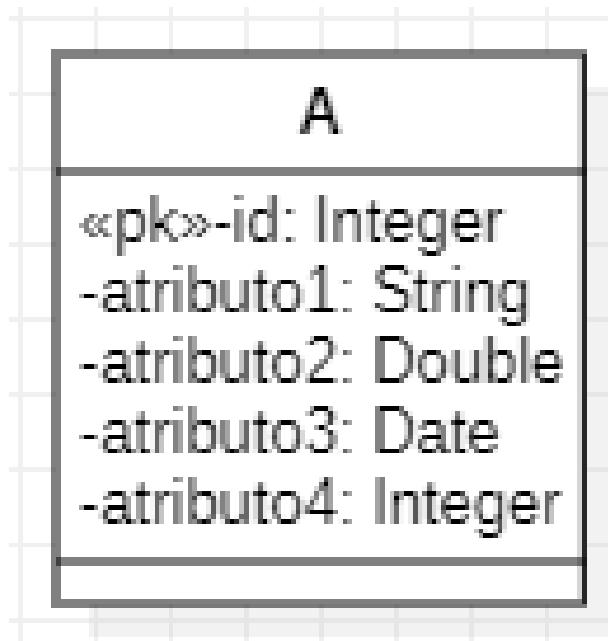
Introdução

- Os serviços que elaboremos obedecerão ao seguinte padrão:
 - Entidades: são classes de domínio do problema (ex: Pessoa, Carro, etc.).
 - Repositórios: são interfaces e classes responsáveis pela persistência. (ex: RepositorioPessoa, RepositorioCarro, etc.).
 - Serviços: são classes responsáveis por interagir com os repositórios e prover funcionalidades para a camada de recursos. (ex: ServicoPessoa, ServicoCarro, etc.).
 - Recursos: são as classes controladoras que efetivamente interagem com “o mundo externo”. (ex: RecursoPessoa, RecursoCarro, etc.).



Criando a primeira Entidade

- Criaremos uma classe A com atributos de tipos “básicos” normalmente utilizados em classes Java:



Criando a primeira Entidade

- Para que o framework funcione corretamente precisaremos fazer algumas implementações básicas:
 - Podemos definir um construtor com parâmetros, mas é obrigatório definir um construtor “vazio”;
 - Devemos gerar os gets e sets para os atributos (para listas criamos apenas os gets);
 - Devemos implementar o hashCode e equals;
 - A classe deve implementar a interface Serializable.



Criando a primeira Entidade

```
import java.io.Serializable;
import java.time.Instant;

public class A implements Serializable{
    private static final long serialVersionUID = 1L;

    private Long id;
    private String atributo1;
    private Double atributo2;
    private Instant atributo3;
    private Integer atributo4;

    public A() {}

    public A(Long id, String atributo1, Double atributo2, Instant atributo3, Integer atributo4) {
        super();
        this.id = id;
        this.atributo1 = atributo1;
        this.atributo2 = atributo2;
        this.atributo3 = atributo3;
        this.atributo4 = atributo4;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
}
```



Criando a primeira Entidade

```
public void setAtributo1(String atributo1) {
    this.atributo1 = atributo1;
}

public Double getAtributo2() {
    return atributo2;
}

public void setAtributo2(Double atributo2) {
    this.atributo2 = atributo2;
}

public Instant getAtributo3() {
    return atributo3;
}

public void setAtributo3(Instant atributo3) {
    this.atributo3 = atributo3;
}

public Integer getAtributo4() {
    return atributo4;
}

public void setAtributo4(Integer atributo4) {
    this.atributo4 = atributo4;
}
```

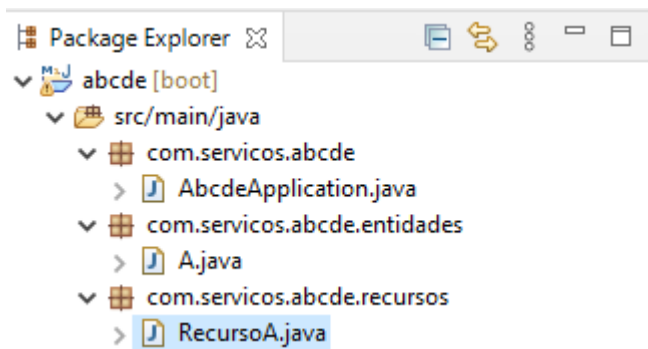
```
@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + ((id == null) ? 0 : id.hashCode());
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    A other = (A) obj;
    if (id == null) {
        if (other.id != null)
            return false;
    } else if (!id.equals(other.id))
        return false;
    return true;
}
```



Criando o RecursoA

- Para conseguirmos usar um recurso a classe deve ser um controlador REST, isso é feito através de uma anotação (**@RestController**).
- Também é necessário definir um caminho para o recurso, isso é feito através de outra anotação (**@RequestMapping(value = "/as")**). Como o recurso está relacionado com a classe A o caminho será **"/as"**.



```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping(value = "/as")
public class RecursoA {

}
```



Criando o RecursoA

- O próximo passo é definir um método que será acessado pelo navegador. Nesse caso, para fins de teste, vamos definir um método que retornaria todos os As (nesse momento vai retornar apenas um A definido “na mão”).

```
import java.time.Instant;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.servicos.abcde.entidades.A;

@RestController
@RequestMapping(value = "/as")
public class RecursoA {

    @GetMapping
    public ResponseEntity<A> obterTodos(){
        A a = new A(1L, "Atributo 1", 7.77, Instant.now(), 7);
        return ResponseEntity.ok().body(a);
    }
}
```



Rodando o serviço web Java

- Para rodar a aplicação faremos o mesmo processo visto na aula passada: Run As → Spring Boot App na classe `AbcdeApplication.java`.



```
abcde - AbcdeApplication [Spring Boot App] C:\sts-4.10.0.RELEASE\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (10 de jun. de 2021 09:44:20)

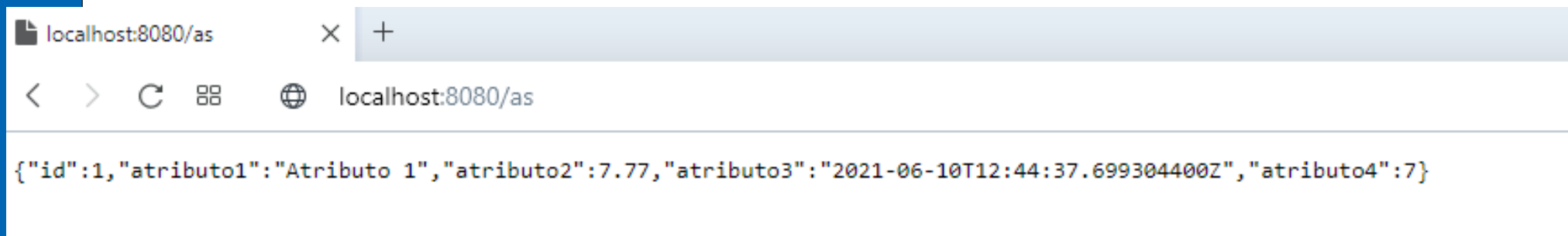
:: Spring Boot :: (v2.5.0)

2021-06-10 09:44:25.382 INFO 848 --- [main] com.servicos.abcde.AbcdeApplication : Starting AbcdeApplication using Java 15.0.2 on DESKTOP-JKB61SJ with PID 848 (C:\Users\Giovany\Documents\SpringToolWorks
2021-06-10 09:44:25.412 INFO 848 --- [main] com.servicos.abcde.AbcdeApplication : No active profile set, falling back to default profiles: default
2021-06-10 09:44:30.496 INFO 848 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2021-06-10 09:44:30.522 INFO 848 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2021-06-10 09:44:30.523 INFO 848 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.46]
2021-06-10 09:44:30.979 INFO 848 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2021-06-10 09:44:30.980 INFO 848 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 5389 ms
2021-06-10 09:44:32.289 INFO 848 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2021-06-10 09:44:32.310 INFO 848 --- [main] com.servicos.abcde.AbcdeApplication : Started AbcdeApplication in 8.043 seconds (JVM running for 10.321)
2021-06-10 09:44:32.312 INFO 848 --- [main] o.s.b.a.ApplicationAvailabilityBean : Application availability state LivenessState changed to CORRECT
2021-06-10 09:44:32.315 INFO 848 --- [main] o.s.b.a.ApplicationAvailabilityBean : Application availability state ReadinessState changed to ACCEPTING_TRAFFIC
2021-06-10 09:44:37.648 INFO 848 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2021-06-10 09:44:37.649 INFO 848 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2021-06-10 09:44:37.650 INFO 848 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
```



Testando o serviço web Java

- Basta ir ao navegador:

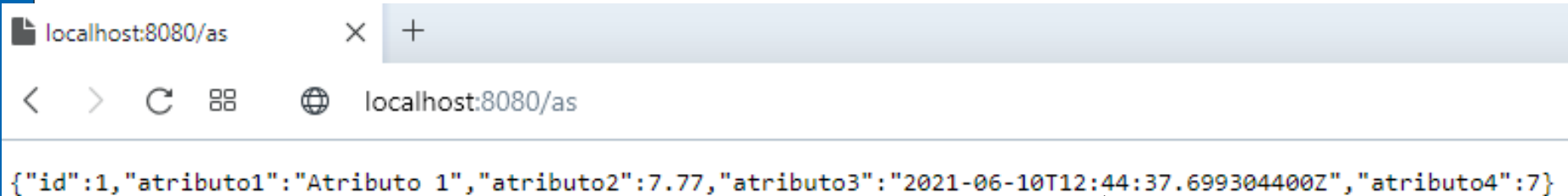


- Notar que o serviço web Java responde no link **localhost:8080/as**. O **/as** foi o caminho definido em **@RequestMapping** na classe **RecursoA**.



Formato da data – ISO 8601

- Vejamos a representação do atributo3 que é do tipo Instant na saída gerada pelo navegador:



```
{ "id":1, "atributo1": "Atributo 1", "atributo2": 7.77, "atributo3": "2021-06-10T12:44:37.699304400Z", "atributo4": 7 }
```

- Essa saída não está no padrão ISO 8601 (veja que os segundos estão com muitos dígitos de representação).



Formato da data – ISO 8601

- Para resolver esse problema utilizaremos a anotação `@JsonFormat`:

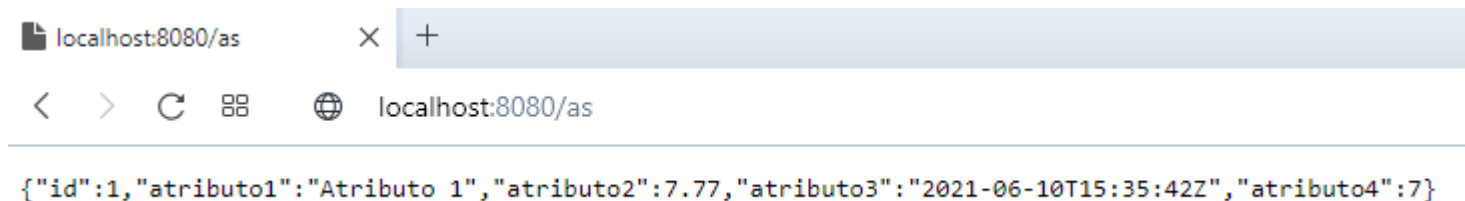
```
import com.fasterxml.jackson.annotation.JsonFormat;

public class A implements Serializable{
    private static final long serialVersionUID = 1L;

    private Long id;
    private String atributo1;
    private Double atributo2;

    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd'T'HH:mm:ss'Z'", timezone = "GMT")
    private Instant atributo3;

    private Integer atributo4;
```



```
localhost:8080/as × +
< > ↺ ☰ 🌐 localhost:8080/as
{"id":1,"atributo1":"Atributo 1","atributo2":7.77,"atributo3":"2021-06-10T15:35:42Z","atributo4":7}
```



Dúvidas?

