

Programmazione Orientata agli Oggetti

Project Lombok su DiaDia

A cura di Ravago e Ursu

Classi modificate

- [Labirinto](#) : Getter , Setter , Builder con LabirintoBuilder
- [CaricatoreLabirinto](#) : Getter su istanza Labirinto e Builder su costruttore
- [ConfigurazioniIniziali](#) : Builder con ConfigurazioniInizialiBuilder senza istanze, con prop modificato
- [Stanza](#) : Getter e Setter, RequiredArgs e Builder su Costruttore
- Varie stanze (Bloccata, Buia, Magica, Protected, MagicaProtected) **non modificate**
- [Attrezzo](#) : Costruttore AllArgs ed annotazione EqualsAndHashCoder ,Getter e Setter
- [AbstractComando](#) : Getter ,Setter (commento secondario)
- Comandi (tutti) **non modificati**
- [Borsa](#) : builder,builder.default , getter
- [Giocatore](#) : Getter, Setter, NoArgsConstructor
- [AbstractPersonaggio](#) : Getter, RequiredArgsConstructor (commento secondario)
- Cane,Mago,Strega **nessuna modifica**

@Builder

@Builder genera il *builder* di una classe.

Lombok crea una classe builder statica interna denominata
NomeClasseAnnotataBuilder.

Es.: Labirinto → LabirintoBuilder

Questa classe builder gestisce l'inizializzazione di tutti i campi della classe annotata con **@Builder**.

Per ogni variabile d'istanza della classe annotata con **@Builder** Lombok genera un metodo del builder che si occupa di impostare il valore di quella specifica variabile. La classe annotata offrirà un metodo statico **builder()** per creare il proprio builder. Il builder offrirà un metodo **build()** per ottenere l'oggetto che si è costruito.

Classe LabirintoBuilder - @Builder con metodi

Senza Lombok

```
public class Labirinto {
    public static LabirintoBuilder newBuilder() {
        return new LabirintoBuilder();
    }
    private Stanza stanzaIniziale;
    private Stanza stanzaVincente;
    public static class LabirintoBuilder {
        private Labirinto labirinto;
        private Stanza ultimaAggiunta;
        private Map<String, Stanza> nome2stanza;

        public LabirintoBuilder() {
            this.labirinto = new Labirinto();
            this.nome2stanza = new HashMap<String, Stanza>();
        }
        public LabirintoBuilder addStanzaIniziale(String nomeStanzaIniziale) { ... }
        public LabirintoBuilder addStanzaVincente(String nomeStanzaVincente) { ... }
        public LabirintoBuilder addAttrezzo(String nomeStanza, String nome, int peso) { ... }
        public LabirintoBuilder addStanza(String nome) { ... }
        public LabirintoBuilder addStanzaMagica(String nome) { ... }
        public Labirinto getLabirinto() {
            return this.labirinto;
        }
    }
}
```

Con Lombok

```
@Builder
public class Labirinto {
    @Getter @Setter private Stanza stanzaIniziale;
    @Getter @Setter private Stanza stanzaVincente;

    public static class LabirintoBuilder {
        private Stanza ultimaAggiunta;
        private Map<String, Stanza> nome2stanza = new HashMap<String, Stanza>();

        public LabirintoBuilder addStanzaIniziale(Stanza stanzaIniziale) { ... }
        public LabirintoBuilder addStanzaVincente(Stanza stanzaVincente) { ... }
        public LabirintoBuilder addAttrezzo(String nomeStanza, String nome, int peso) { ... }
        public LabirintoBuilder addStanza(Stanza stanza) { ... }
        public LabirintoBuilder addStanzaMagica(String nome) { ... }
        ...
    }
}
```

Creazione di un Labirinto tramite Builder

```
Labirinto labirinto = Labirinto.builder()
    .addStanzaIniziale(labCampus)
    .addStanzaVincente(biblioteca)
    .build();
```

Caricatore Labirinto - @Builder su Costruttore

prima senza Lombok

```
public class CaricatoreLabirinto {
    private final String STANZE = "Stanze:";
    private final String ATTREZZI = "Attrezzi:";
    private final String USCITE = "Uscite:";
    private final String ESTREMI = "Estremi:";
    private final String MAGICHE = "Magiche:";
    private BufferedReader reader;
    private Set<String> stanze;
    private int numeroLinea;
    private LabirintoBuilder builder;

    public CaricatoreLabirinto(String nomeFile) throws
FileNotFoundException {
        this(new FileReader(nomeFile));
    }

    public CaricatoreLabirinto(Reader reader) {
        this.stanze = new HashSet<String>();
        this.numeroLinea = 0;
        this.reader = new BufferedReader(reader);
        this.builder = Labirinto.newBuilder();
    }
    ...
}
```

dopo con Lombok

```
public class CaricatoreLabirinto {
    private final String STANZE = "Stanze:";
    private final String ATTREZZI = "Attrezzi:";
    private final String USCITE = "Uscite:";
    private final String ESTREMI = "Estremi:";
    private final String MAGICHE = "Magiche:";
    private BufferedReader reader;
    private Set<String> stanze = new HashSet<String>();
    private int numeroLinea = 0;
    private LabirintoBuilder builder = Labirinto.builder();
    @Getter private Labirinto labirinto;

    public CaricatoreLabirinto(String nomeFile) throws
FileNotFoundException {
        this(new FileReader(nomeFile));
    }
    @Builder
    public CaricatoreLabirinto(Reader reader) {
        this.reader = new BufferedReader(reader);
    }
    public void carica() {
        ....
        labirinto= builder.build();
        ....
    }
}
```

Classe Borsa - @Builder(con Default) e @Getter

Se una determinata istanza non viene mai impostata durante una compilazione, ottiene sempre **0 / null / false**.

Se si mette **@Builder** su una classe (e non su un metodo o un costruttore) è possibile specificare il valore predefinito direttamente sull'istanza e annotare l'istanza con **@Builder.Default**:

@Builder

```
public class Borsa {
```

```
    public final static int DEFAULT_PESO_MAX_BORSA = ConfigurazioneIniziali.getPesoMax();
```

```
    @Builder.Default private Map<String, Attrezzo> nome2attrezzo = new HashMap<>();
```

```
    @Builder.Default @Getter private int pesoMax = DEFAULT_PESO_MAX_BORSA;
```

```
    @Getter private int pesoAttuale;
```

```
}
```

Classe Borsa - @Builder.Default (1)

il Builder della classe Borsa si ottiene mettendo Borsa.builder() :

```
public class BorsaTest {

    private Borsa borsa;
    private Borsa borsa2;
    private static final int PESO_MAX_BORSA = 5;

    @Before
    public void setUp() {
        this.borsa = Borsa.builder().pesoMax(PESO_MAX_BORSA).build();
        this.borsa2 = Borsa.builder().build();
    }

    @Test
    public void testGetPesoMax() {
        assertEquals(PESO_MAX_BORSA, this.borsa.getPesoMax()); // pesoMax borsa inizializzato con builder
        assertEquals(10, this.borsa2.getPesoMax()); // pesoMax borsa di default = 10
    }
}
```

Classe ConfigurazioniIniziali - @Builder

@Builder

```
public final class ConfigurazioniIniziali {  
    private static final String DIADIA_PROPERTIES = "diadia.properties";  
    private static final String PESO_MAX = "pesoMax";  
    private static final String CFU = "cfu";  
    private static Properties prop = null;  
    public static class ConfigurazioniInizialiBuilder {  
        public ConfigurazioniInizialiBuilder cfulniziali(int cfulniziali) {  
            if(prop == null) carica();  
            prop.setProperty(CFU,String.valueOf(cfulniziali));  
            return this;  
        }  
        public ConfigurazioniInizialiBuilder pesoMaxBorsa(int pesoMaxBorsa) {  
            if(prop == null) carica();  
            prop.setProperty(PESO_MAX,String.valueOf(pesoMaxBorsa));  
            return this;  
        }  
    }  
}
```

Utilizzo del builder per le ConfigurazioniIniziali che sovrascrivono i valori di default nel diadia.properties

```
ConfigurazioniIniziali config = ConfigurazioniIniziali.builder().cfulniziali(20).pesoMaxBorsa(30).build();
```


@Getter & @Setter

Le annotazioni **@Getter** e **@Setter** su una variabile di istanza di nome X generano i metodi **getX()** e **setX()**

- **isX()** invece di **getX()** se **boolean**
- i metodi generati sono sempre **public**

Può però essere specificato un **AccessLevel** alternativo: **PUBLIC**, **PROTECTED**, **PACKAGE**, **PRIVATE**, e **NONE**


@Setter(AccessLevel.PROTECTED)

Classe AbstractComando - @Getter,@Setter

```
public abstract class AbstractComando {  
  
    private IO io;  
    private String nome;  
    private String parametro;  
  
    public abstract void esegui(Partita partita);  
  
    protected IO getIO() { return this.io; }  
    public void setIO(IO io) { this.io = io; }  
    public String getNome() { return this.nome; }  
    protected void setNome(String nome) { this.nome = nome; }  
    public String getParametro() { return this.parametro; }  
    public void setParametro(String parametro) {this.parametro = parametro;}  
  
}
```

```
public abstract class AbstractComando {  
  
    @Getter @Setter private IO io;  
    @Getter @Setter private String nome;  
    @Getter @Setter private String parametro;  
  
    public abstract void esegui(Partita partita);  
  
}
```

 AbstractComando.java
▼  AbstractComando

- ▣ io
- ▣ nome
- ▣ parametro
-  esegui(Partita) : void
-  getIO() : IO
-  getNome() : String
-  getParametro() : String
-  setIO() : void
-  setNome(String) : void
-  setParametro(String) : void

Classe Giocatore - @Getter & @Setter (1)

```
public class Giocatore {  
  
    static final public int CFU_INIZIALI = ConfigurazioniIniziali.getCFU();  
  
    @Getter @Setter private int cfu = CFU_INIZIALI;  
    @Getter private Borsa borsa = Borsa.builder().build();  
  
}
```



Classe Giocatore - @NoArgsConstructor


@NoArgsConstructor genera un costruttore senza parametri.

Senza Lombok

```
public class Giocatore {  
  
    static final public int CFU_INIZIALI = ConfigurazioniIniziali.getCFU();  
  
    private int cfu = CFU_INIZIALI;  
    private Borsa borsa;  
  
    public Giocatore() {  
        this.cfu = CFU_INIZIALI;  
        this.borsa = Borsa.builder().build();  
    }  
}
```

Con Lombok

```
@NoArgsConstructor  
public class Giocatore {  
  
    static final public int CFU_INIZIALI = ConfigurazioniIniziali.getCFU();  
  
    @Getter @Setter private int cfu = CFU_INIZIALI;  
    @Getter private Borsa borsa = Borsa.builder().build();  
}
```



Classe Stanza - @RequiredArgsConstructor

La notazione **@RequiredArgsConstructor** genera un costruttore con tutti i parametri necessari per una completa inizializzazione dei nuovi oggetti, ovvero, un parametro per ciascun variabile d'istanza:

- dichiarata **final** e non inizializzata
- marcata con l'annotazione **@NonNull**

Classe Stanza - @RequiredArgsConstructor

Senza Lombok

```
public class Stanza {
    static final private int NUMERO_MASSIMO_DIREZIONI = 4;
    static final public int NUMERO_MASSIMO_ATTREZZI = 10;
    private String nome;
    protected Map<String, Attrezzo> nome2attrezzo;
    private Map<Direzione, Stanza> direzione2stanzaAdiacente;
    private AbstractPersonaggio personaggio;

    public String getNome() {
        return this.nome;
    }
    public void setPersonaggio(AbstractPersonaggio personaggio) {
        this.personaggio = personaggio;
    }
    public AbstractPersonaggio getPersonaggio() {
        return this.personaggio;
    }
    public Stanza(String nome) {
        this.nome = nome;
        this.nome2attrezzo = new HashMap<>();
        this.direzione2stanzaAdiacente = new HashMap<>();
    }
    ...
}
```

Con Lombok

```
@RequiredArgsConstructor
public class Stanza {

    static final private int NUMERO_MASSIMO_DIREZIONI = 4;
    static final public int NUMERO_MASSIMO_ATTREZZI = 10;

    @NonNull @Getter private String nome;
    protected Map<String, Attrezzo> nome2attrezzo = new HashMap<>();
    private Map<Direzione, Stanza> direzione2stanzaAdiacente = new HashMap<>();
    @Getter @Setter private AbstractPersonaggio personaggio;

    ...
}
```



Classe AbstractPersonaggio - @RequiredArgsConstructor


Senza Lombok

```
public abstract class AbstractPersonaggio {  
  
    private String nome;  
    private String presentazione;  
    private boolean haSalutato;  
  
    public AbstractPersonaggio(String nome, String presentaz) {  
        this.nome = nome;  
        this.presentazione = presentaz;  
        this.haSalutato = false;  
    }  
}
```

Con Lombok

```
@RequiredArgsConstructor  
public abstract class AbstractPersonaggio {  
  
    @NonNull @Getter private String nome;  
    @NonNull private String presentazione;  
    @Getter private boolean presentato = false;  
}
```

 AbstractPersonaggio.java
▼  AbstractPersonaggio

- ▣ nome
- ▣ presentato
- ▣ presentazione
-  AbstractPersonaggio(String, String)

Classe Attrezzo - @AllArgsConstructor

La notazione **@AllArgsConstructor** genera un costruttore con un parametro per ogni variabile d'istanza della classe, di default il costruttore generato sarà di tipo public.

Senza Lombok

```
public class Attrezzo implements Comparable<Attrezzo> {  
  
    private String nome;  
    private int peso;  
  
    public Attrezzo(String nome, int peso) {  
        this.peso = peso;  
        this.nome = nome;  
    }  
}
```

Con Lombok

```
@AllArgsConstructor  
public class Attrezzo implements Comparable<Attrezzo> {  
  
    @Getter private String nome;  
    @Getter @Setter private int peso;  
  
    ...  
}
```



@EqualsAndHashCode

@EqualsAndHashCode per generare automaticamente i metodi **equals()** e **hashCode()**

Senza Lombok

```
public class Attrezzo implements Comparable<Attrezzo> {  
  
    @Getter private String nome;  
    @Getter @Setter private int peso;  
    ...  
  
    @Override  
    public boolean equals(Object obj) {  
        Attrezzo that = (Attrezzo)obj;  
        return this.getNome().equals(that.getNome()) && this.getPeso() == that.getPeso();  
    }  
  
    @Override  
    public int hashCode() {  
        return this.getNome().hashCode() + this.getPeso();  
    }  
    ...  
}
```

Con Lombok

```
@EqualsAndHashCode  
public class Attrezzo implements Comparable<Attrezzo>  
{  
  
    @Getter private String nome;  
    @Getter @Setter private int peso;  
    ...  
}
```

