Advanced EE6 Lab

Part 3, CDI Portable Extensions and DeltaSpike

Marek Schmidt

Dec 2012

Project

- git clone git://github.com/qa/pv243.git
- cd pv243
- git checkout deltaspike-00
- JBDS -> Import... -> Maven -> Existing Maven Projects -> lesson03-cdi-pe
- http://wumpus-social.rhcloud.com

Wumpus Social

- Based on the original BSD game “wumpus”

Task 0

- Explore the application
- room.xhtml / RoomAction
- CurrentPlayerManager
- RoomEventsNarrator
- Player / Room
- EntryRoom, EastRoom, SouthRoom, PitRoom

Task 1 Make it more social

- Change the application so there is only one set of room instances, so all the players would share the rooms

Task 1 Solution

- Change the rooms scope from @SessionScoped to @ApplicationScoped

Task 2 Configure the rooms from a xml file

- Write a CDI extension which would configure Room beans from a xml file
- (ignore the room description and smell for now)

- ```
  class Room {
    private Room north;
    private Room south;
    ..
  }
  ```

```
@RoomName(“room1”)
@ApplicationScoped
class Room {
   @Inject @RoomName(“room2”)
   private Room north;

   ...
}
```

```
@RoomName(“room2”)
@ApplicationScoped
class Room {
   @Inject @RoomName(“room1”)
   private Room south;

   ...
}
```



```

Task 2 Hints

- XmlRoomBeansExtension
- META-INF/services/javax.enterprise.inject.spi.Extension
- BeforeBeanDiscovery event, addAnnotatedType(...)
- DeltaSpike AnnotatedTypeBuilder
  - readFromType
  - addToClass
  - addToField
  - create
- class InjectLiteral extends AnnotationLiteral<Inject> implements Inject {}

Task 3 Room description and smell

- Create @StringsEntry qualifier, producer and add @Inject @StringsEntry annotations into description and smell fields of our room annotated types in XmlRoomBeansExtension
- ```
  @ApplicationScoped
  @RoomName(“foo”)
  class Room {
    @Inject
    @StringsEntry(“You fall into a bottomless pit”)
    private String description;

    @Inject
    @StringsEntry(“You feel a breeze”)
    private String smell;
    ...
  }
  ```
- ```
  @Qualifier
  @Target({ FIELD })
  @Retention(RUNTIME)
  public @interface StringsEntry {
    @Nonbinding String value() default "";
  }
  ```


Task 3 Hints

```
public class StringsProducer {
  @Produces
  @StringsEntry
  public String getString(InjectionPoint ip) {
    StringsEntry annotation =
ip.getAnnotated().getAnnotation(StringsEntry.class);
    return annotation.value();
  }
}
```

- Continue with the XmlRoomBeansExtension addRoom method
- (DeltaSpike)

  AnnotationInstanceProvider.of(StringsEntry.class, values);
- AnnotatedTypeBuilder addToField
- new InjectLiteral()

Task 4 @GameScoped

- Create a custom context GameContext
- Each game have a Integer identifier
- A bean may produce a @CurrentGameId Integer to specify the current game context.
- Game contexts will be controlled by a @ApplicationScoped GamesManager bean

Task 4 Hints

- GameScopeExtension
  - META-INF/services/javax.enterprise.inject.spi.Extension
  - AfterBeanDiscovery addContext
- GamesManager
  - Uncomment GameContext calls
- GameContext
  - Implement getCurrentGameId
    - beanManager.getBeans
    - beanManager.createCreationalContext
    - beanManager.getReference
    - bean.destroy
- XmlRoomBeansExtension GameContextLiteral

Task 5 Random Room Producer

- Create a random Room Producer
- Modify the CurrentPlayerManager to inject a random initial room.

Task 5 Hints

- @ApplicationScoped RandomRoomProducer
  - @Produces @Random Room getRandomRoom() {...}
- private java.util.Random random = new java.util.Random();
- 1. get the room names available by reading the RoomName qualifiers on all the Room-typed beans
  - beanManager.getBeans(Room.class, new AnyLiteral());
  - Bean getQualifiers
- 2. get a random RoomName qualifier
- 3. @Inject @Any Instance<Room> roomInstance;

```
roomInstance.select(
  new RoomName.RoomNameLiteral(randomName));
```


Task 6 Add Wumpus

- Create a @GameScoped Wumpus bean
- Wumpus sits in a random room, stays there and eats any traveller that enters his room
- Wumpus smells really bad (2 rooms)
- Player that shoots at the room in which Wumpus sits, wins and all the other players lose.
  - onMove(@Observes PlayerEnteredRoomEvent event, ...
  - onShoot(@Observes PlayerShootAtRoomEvent event, ...

Task 6 Hints

- Smelling means gameMessage.add() when a player moves to a room that is near the room wumpus sits.
- Winning/ending the game is the same as killing a player
  - currentPlayer.setAlive(false);
  - currentPlayer.setDeathMessage("You have killed the wumpus! You won!");
- You can inject a list of all the players in the current game
  - @Current List<Player> players

Task 7 Ordered Observers

- Create an extension that would allow ordering of event observers
  - roomDescriptionObserver(@Observes @ObserverOrder(0) PlayerEnteredRoomEvent event, ...)
  - roomSmellObserver(@Observes @ObserverOrder(1) PlayerEnteredRoomEvent event, ...)
- @ObserverOrder would be a qualifier

```
orderingObserver(@Observer PlayerEnteredRoomEvent event) {
  roomDescriptionObserverMethod.notify(event);
  roomSmellObserverMethod.notify(event);
}
```


Task 7 Hints

- register ObserverOrderExtension
- Implement the processObserverMethod to gather the observer methods with the @ObserverOrder qualifier

The End