EAPLI

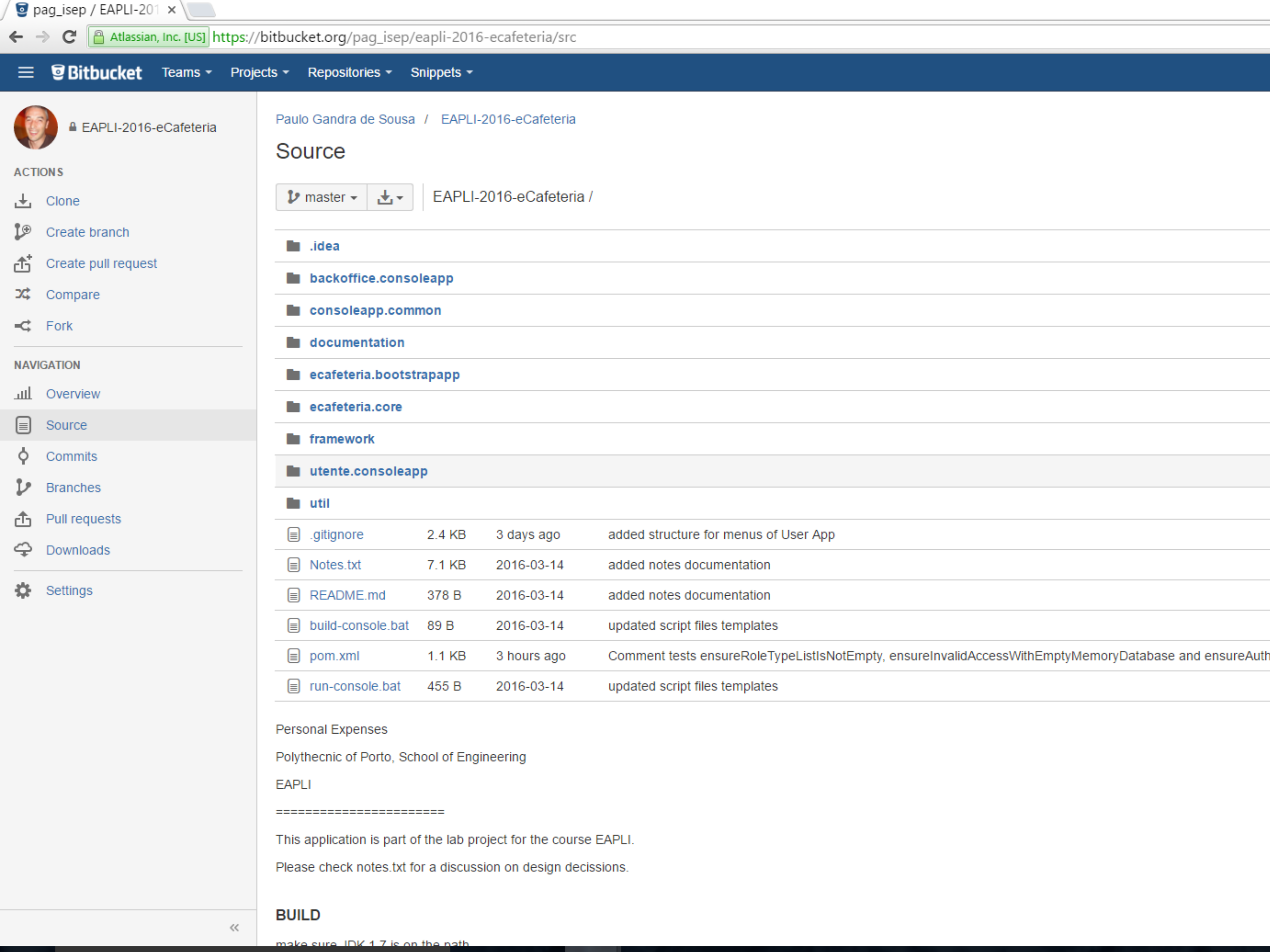
Projeto PL Base eCafeteria

eCafeteria

- Cafeteria management
- Users, Kitchen and Menu managers, Cashiers
- User app
- Backoffice app
    - Kitchen management
    - Menu management
    - Delivery station

Scope of this lesson

- Project structure
- Architecture (same as proposed in classes)
- No business discussion
- Overview of existing codebase
- Focus on “User” and “Dish Type” domain concepts
- Two use cases
  - Register a new user
  - List all dish types



Source

EAPLI-2016-eCafeteria /

.idea			
backoffice.consoleapp			
consoleapp.common			
documentation			
ecafeteria.bootstrapapp			
ecafeteria.core			
framework			
utente.consoleapp			
util			
.gitignore	2.4 KB	3 days ago	added structure for menus of User App
Notes.txt	7.1 KB	2016-03-14	added notes documentation
README.md	378 B	2016-03-14	added notes documentation
build-console.bat	89 B	2016-03-14	updated script files templates
pom.xml	1.1 KB	3 hours ago	Comment tests ensureRoleTypeListIsNotEmpty, ensureInvalidAccessWithEmptyMemoryDatabase and ensureAuth
run-console.bat	455 B	2016-03-14	updated script files templates

Personal Expenses

Polythecnic of Porto, School of Engineering

EAPLI

=======================

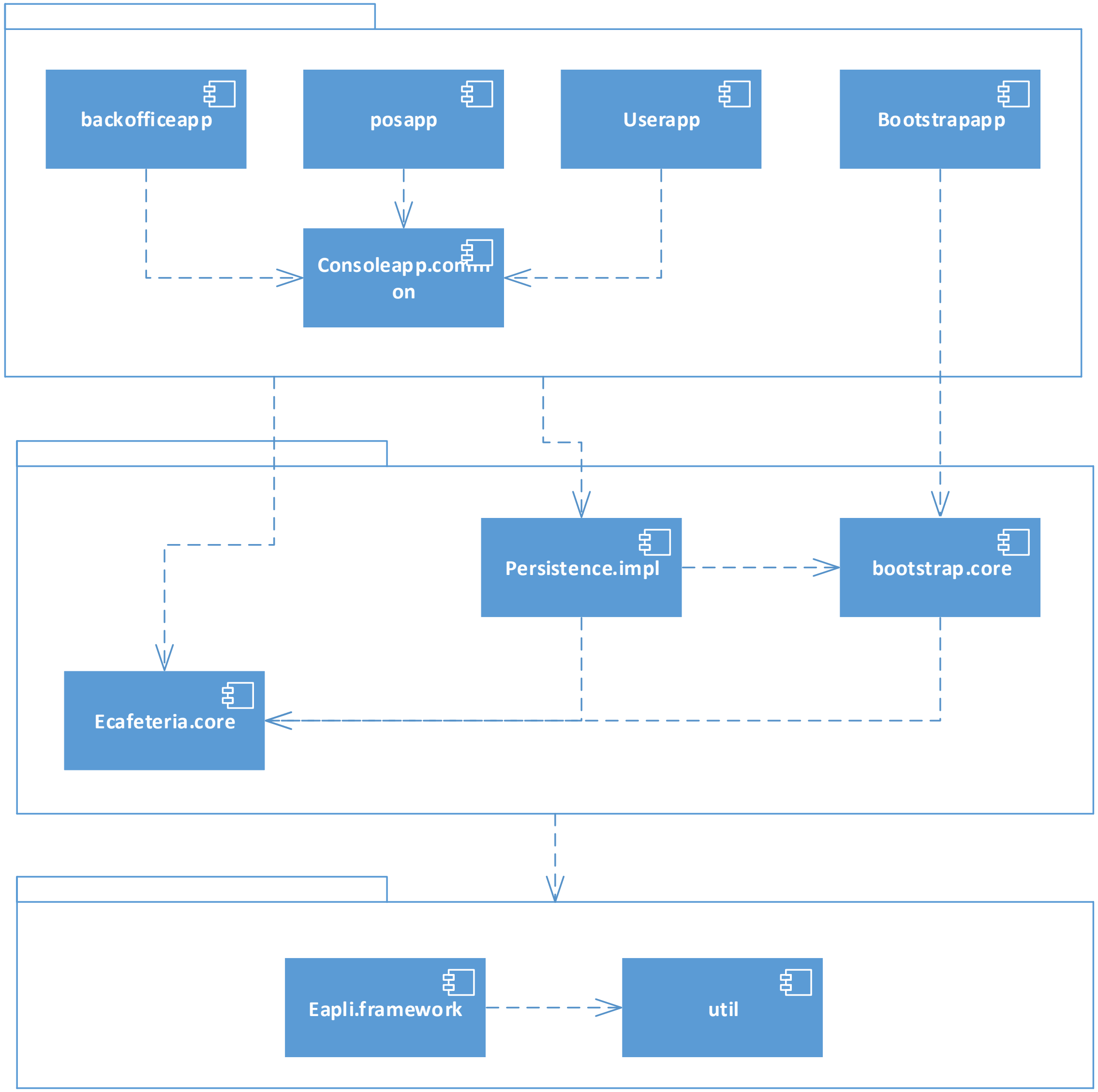
This application is part of the lab project for the course EAPLI.

Please check notes.txt for a discussion on design decissions.

BUILD

make sure JDK 1.7 is on the path

Components (a.k.a. projects)



- Backofficeapp, userapp, pos
- Core, console.common
- Persistence.impl
- bootstrap

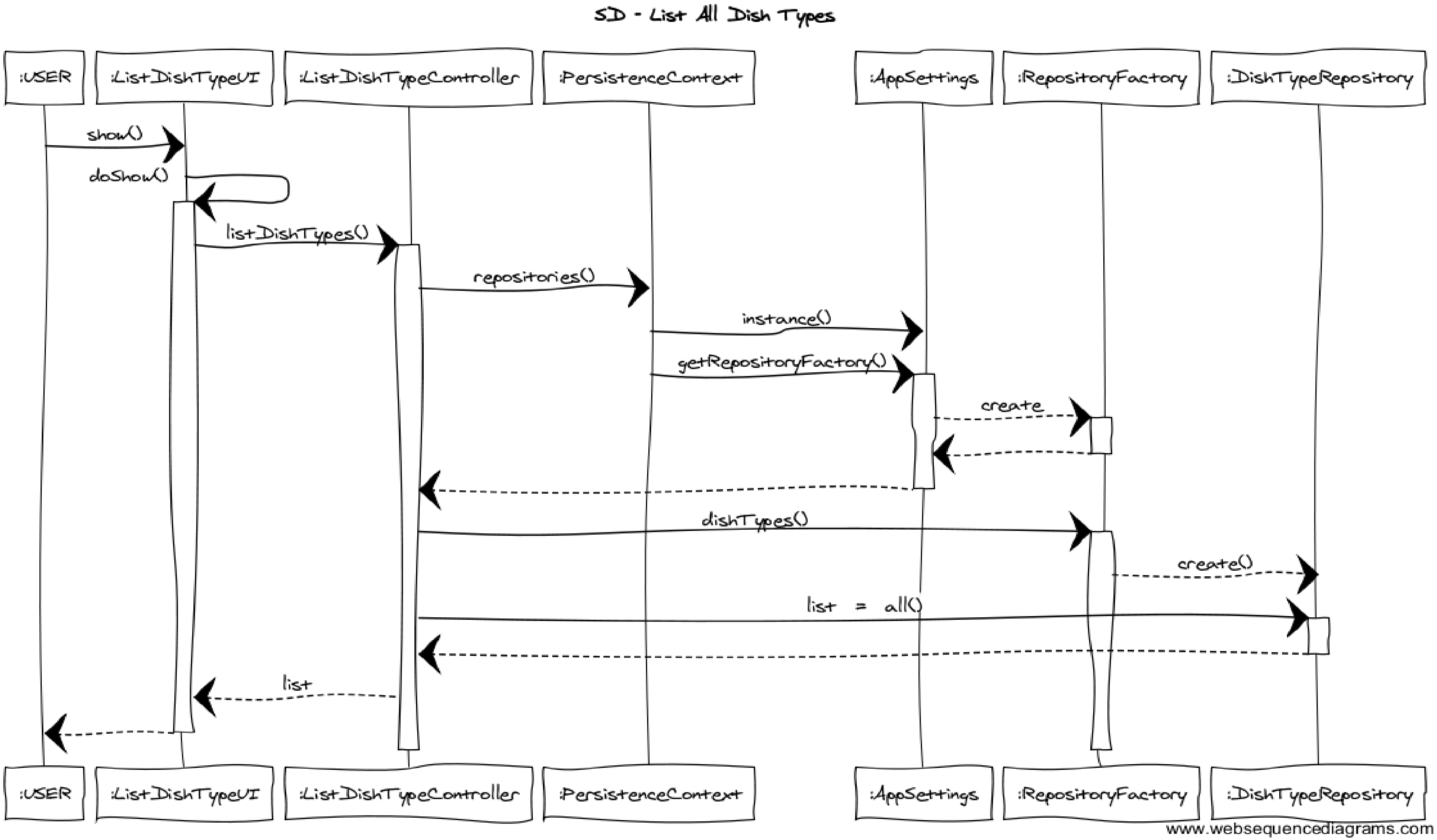
- Framework
  - Utility classes for DDD applications with JPA in EAPLI context
- Util
  - Generic utility classes

eCafeteria design decisions

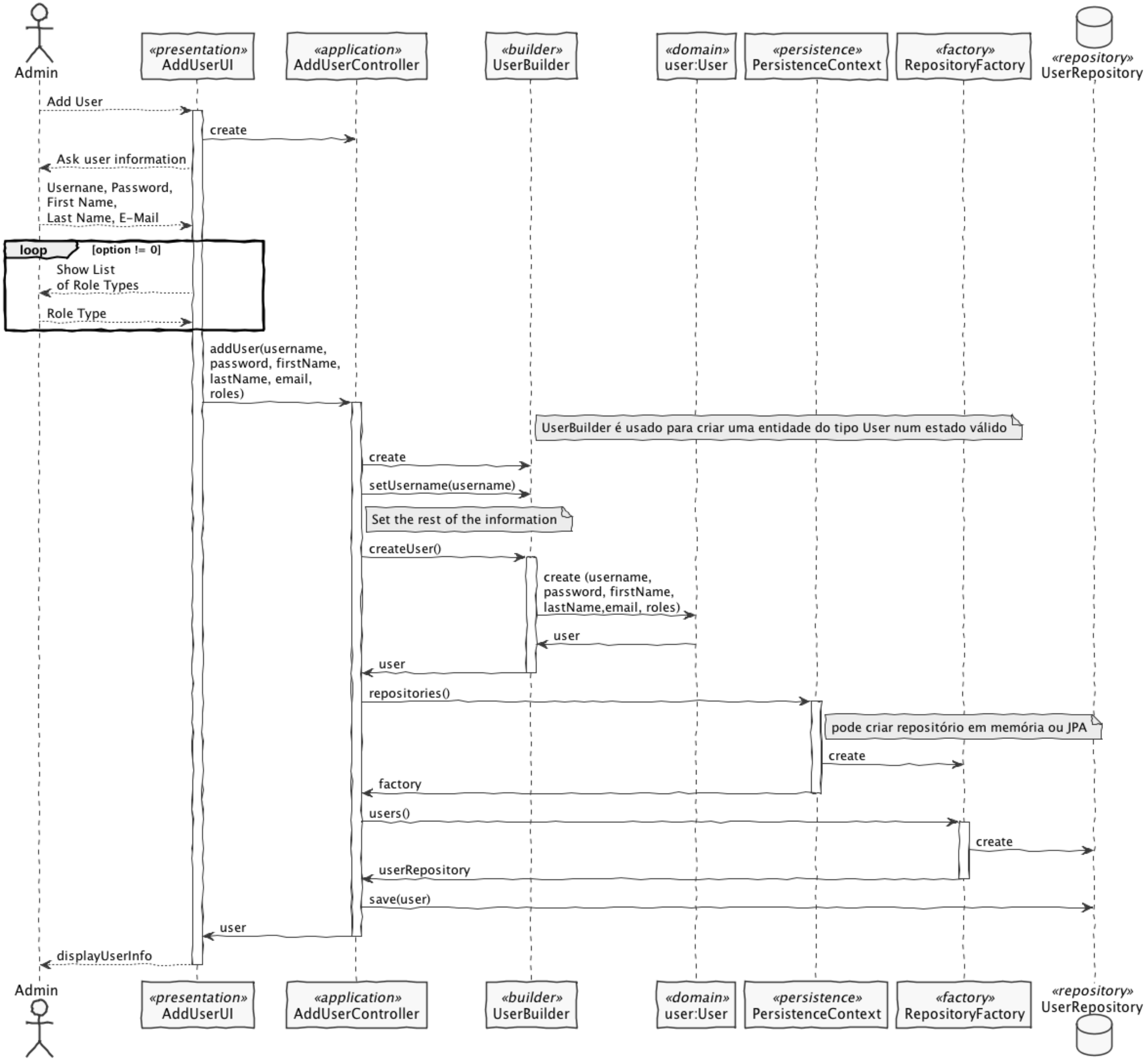
- Layers
  - Presentation
  - Application
  - Domain
  - Persistence
- Domain objects travel to UI for output

More on this next lesson

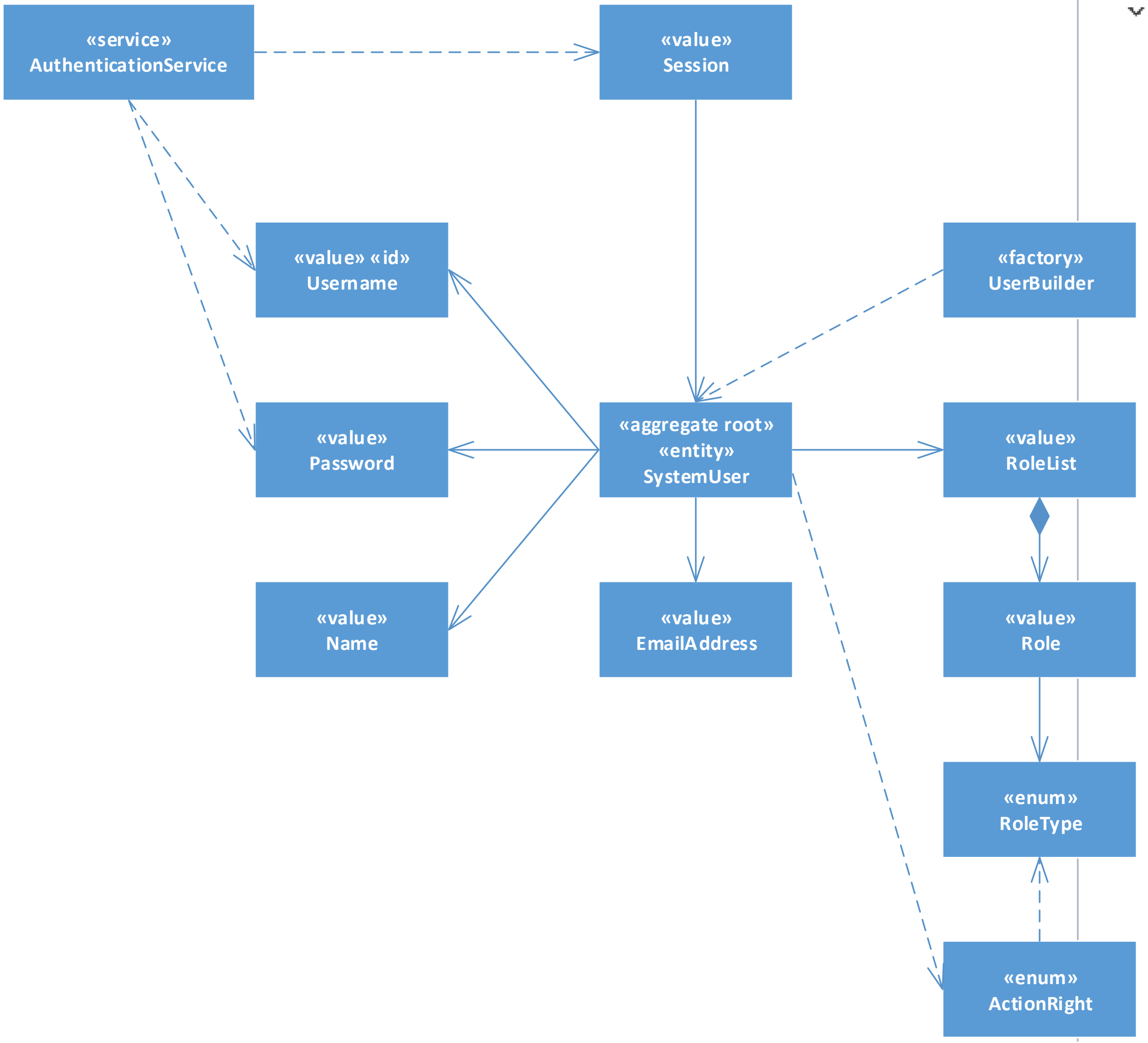
List Dish Types

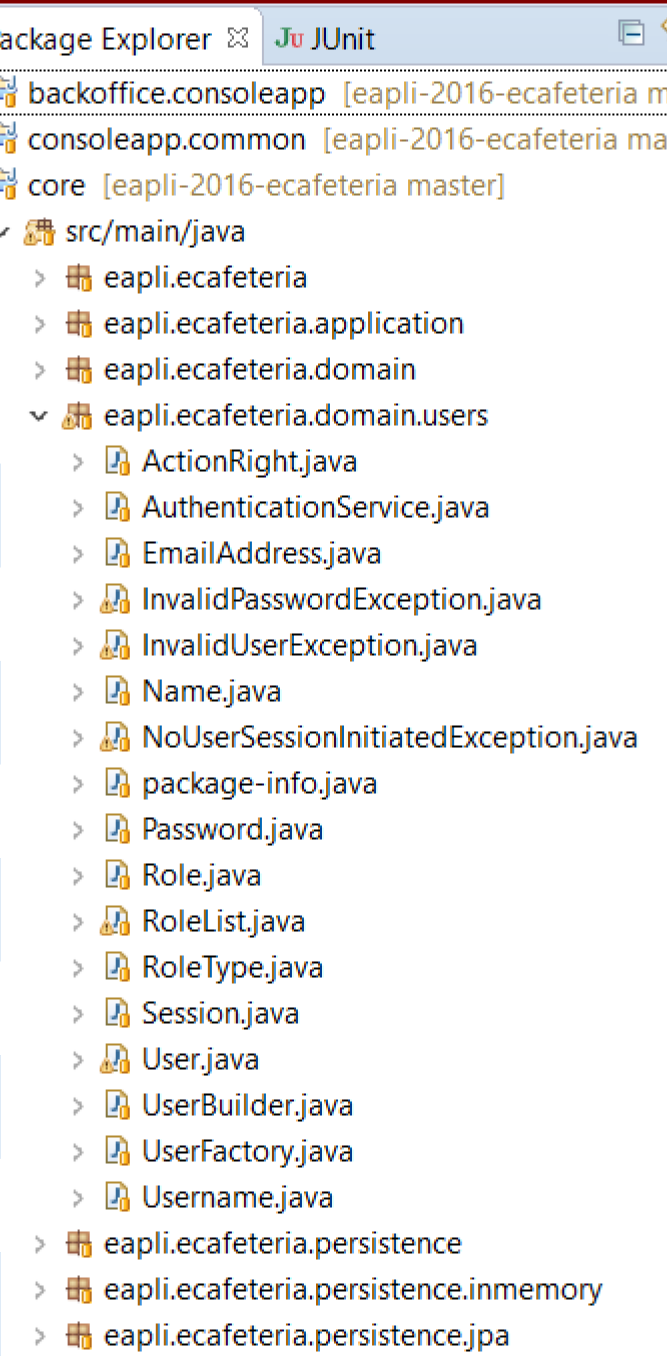


Register New User



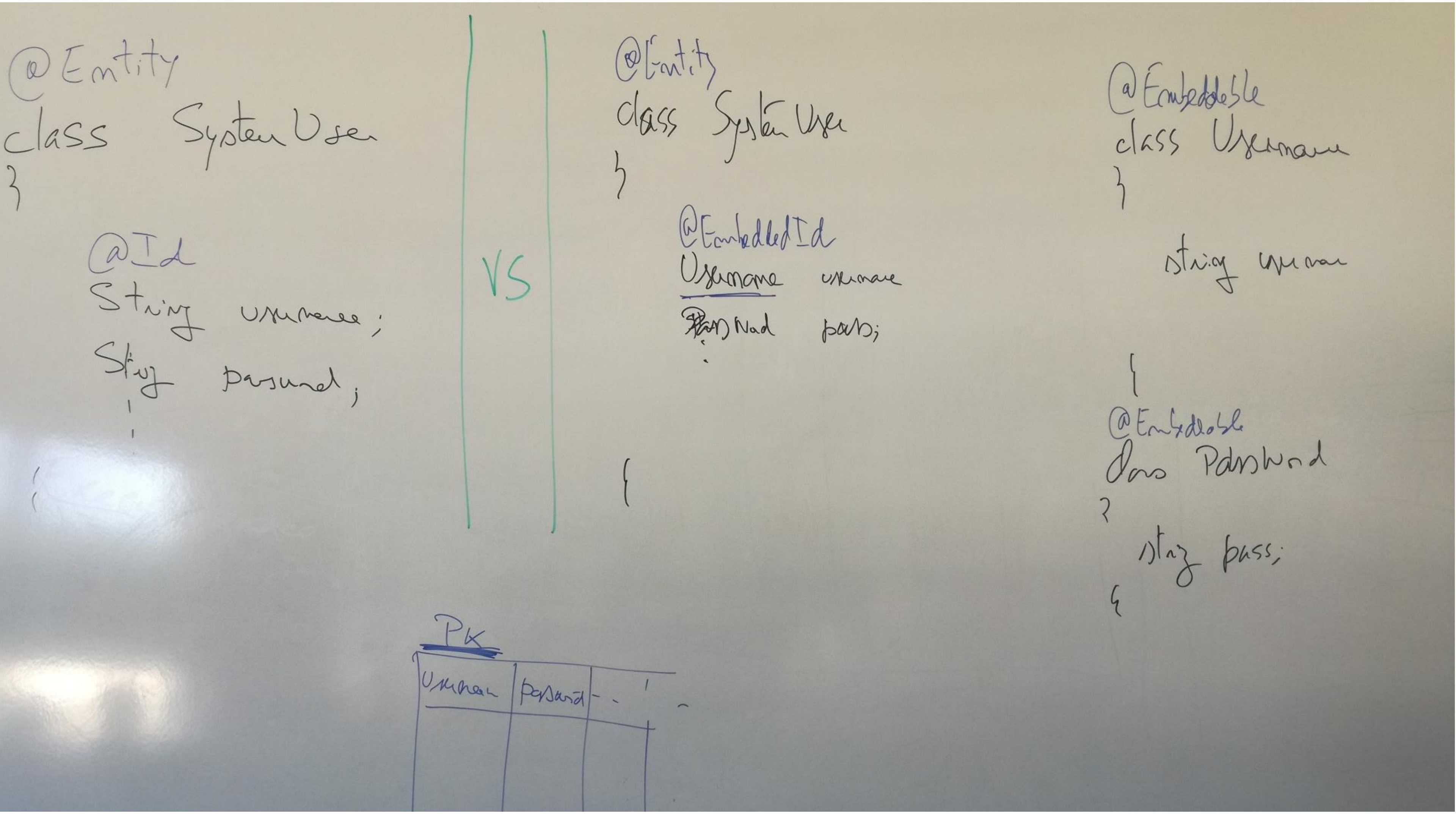
User model





- backoffice.consoleapp [eapli-2016-ecafeteria m
- consoleapp.common [eapli-2016-ecafeteria ma
- core [eapli-2016-ecafeteria master]
  - src/main/java
    - eapli.ecafeteria
    - eapli.ecafeteria.application
    - eapli.ecafeteria.domain
    - eapli.ecafeteria.domain.users
      - ActionRight.java
      - AuthenticationService.java
      - EmailAddress.java
      - InvalidPasswordException.java
      - InvalidUserException.java
      - Name.java
      - NoUserSessionInitiatedException.java
      - package-info.java
      - Password.java
      - Role.java
      - RoleList.java
      - RoleType.java
      - Session.java
      - User.java
      - UserBuilder.java
      - UserFactory.java
      - Username.java
    - eapli.ecafeteria.persistence
    - eapli.ecafeteria.persistence.inmemory
    - eapli.ecafeteria.persistence.jpa

Embeddable vs regular fields



```
@Entity
class SystemUser
}
    @Id
    String username;
    String password;
    .
    .
{
```

VS

```
@Entity
class SystemUser
}
    @EmbeddedId
    Username username
    Password pass;
    .
{
```

```
@Embeddable
class Username
}
    String username
{
```

```
@Embeddable
class Password
}
    String pass;
{
```

PK

Username	Password	- -

Domain invariants

```
@Test
public void ensurePasswordHasAtLeastOneDigitAnd6CharactersLong()
{
        new Password("abcdefgh1");
}

@Test(expected = IllegalArgumentException.class)
public void ensurePasswordsSmallerThan6CharactersAreNotAllowed()
{
        new Password("ab1c");
}

@Test(expected = IllegalArgumentException.class)
public void ensurePasswordsWithoutDigitsAreNotAllowed() {
        new Password("abcdefgh");
}
```

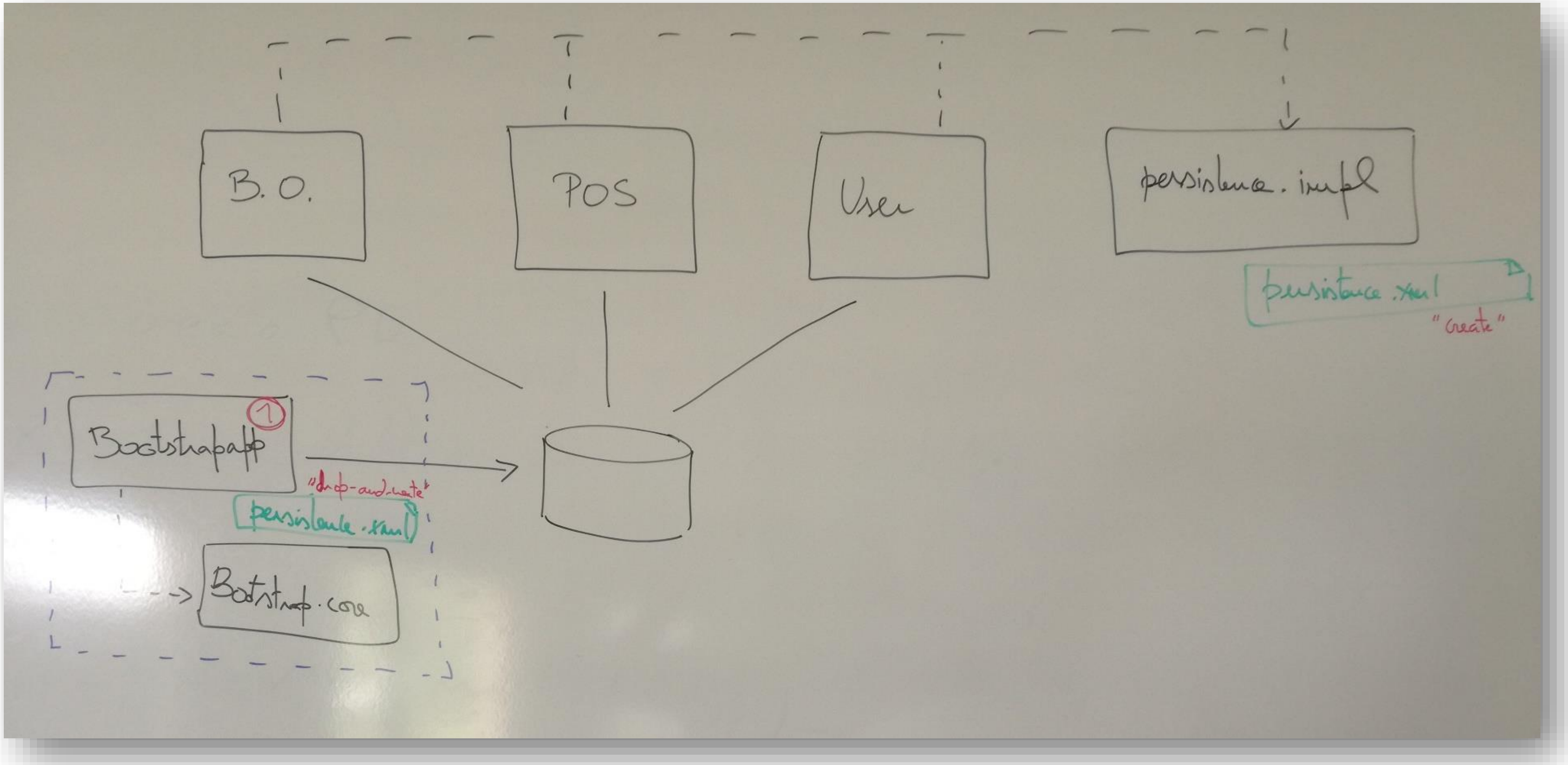
Domain

```
public class Password {

…

public Password(String password) {
        if (!meetsMinimumRequirements(password)) {
            throw new IllegalStateException();
        }
        thePassword = password;
}

private boolean meetsMinimumRequirements(String password) {
        if (Strings.isNullOrEmpty(password)
        || password.length() < 6)
        || !Strings.containsDigit(password))
        {
            return false;
        }

        return true;
}
```

Some additional design decisions

- Bootstrap data
- Support two repositories
  - In memory
  - Relational database
- Decide which repository implementation to use based on property file
- Simple main menu

Bootstrap



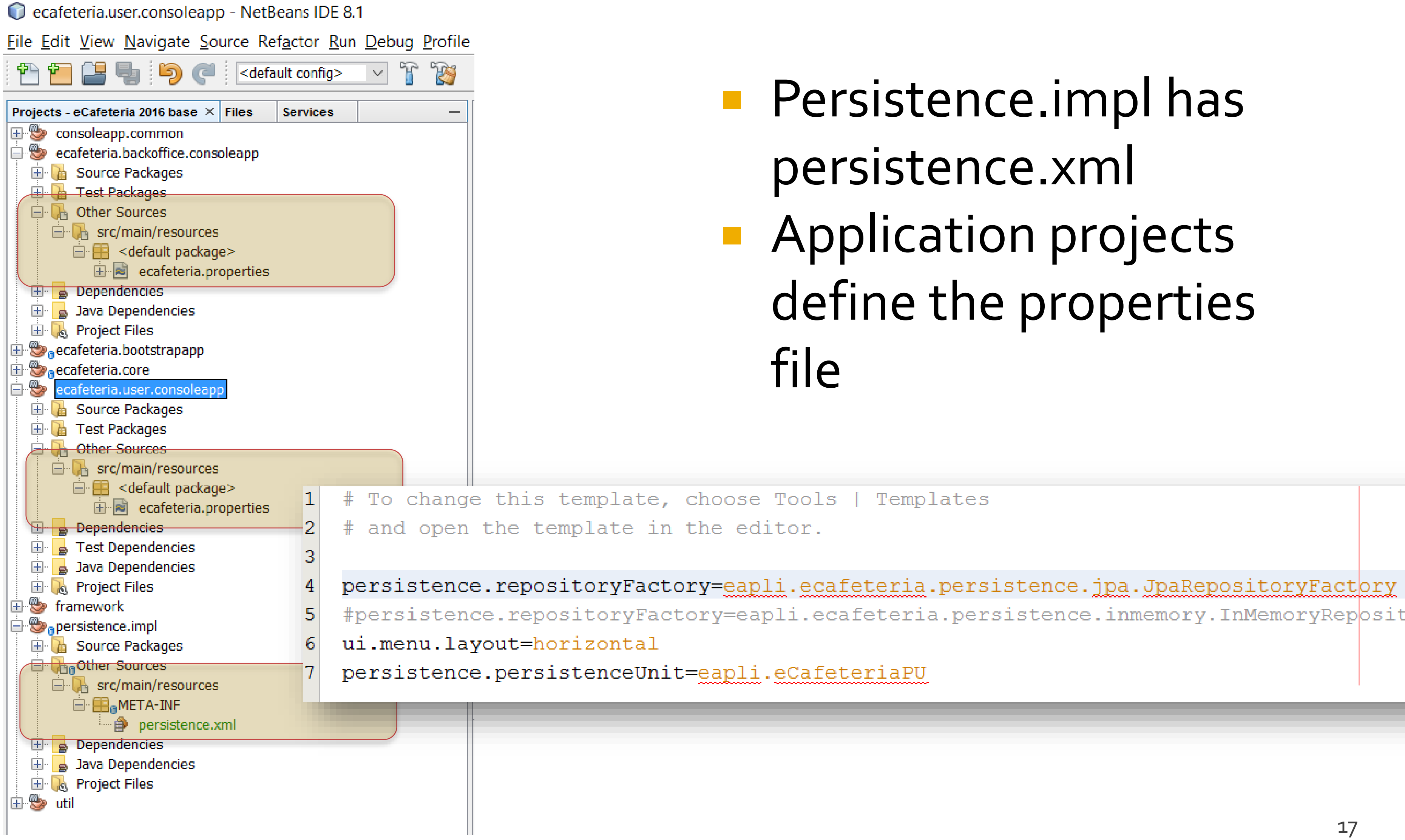
Separate bootstrapapp for database inicialization

bootstrap

```
public class ECafeteriaBootstraper implements Action {

        @Override
        public boolean execute() {
                // declare bootstrap actions
                final Action[] actions = { new UsersBootstrap(), };
                // execute all bootstrapping
                boolean ret = false;
                for (final Action boot : actions) {
                        ret |= boot.execute();
                }
                return ret;
        }
}

public class UsersBootstrap implements Action {

        @Override
        public boolean execute() {
                registerAdmin();
                return false;
        }

        private void registerAdmin() {
                final String username = "admin";
                final String password = "admin";
                final String firstName = "John";
                final String lastName = "Doe";
                final String email = "john.doe@emai.l.com";
                final List<RoleType> roles = new ArrayList<RoleType>();
                roles.add(RoleType.Admin);

                final UserRegisterController userController = new UserRegisterController();
                userController.registerUser(username, password, firstName, lastName, email, roles);
        }
}
```

Resources

- Persistence.impl has persistence.xml
- Application projects define the properties file

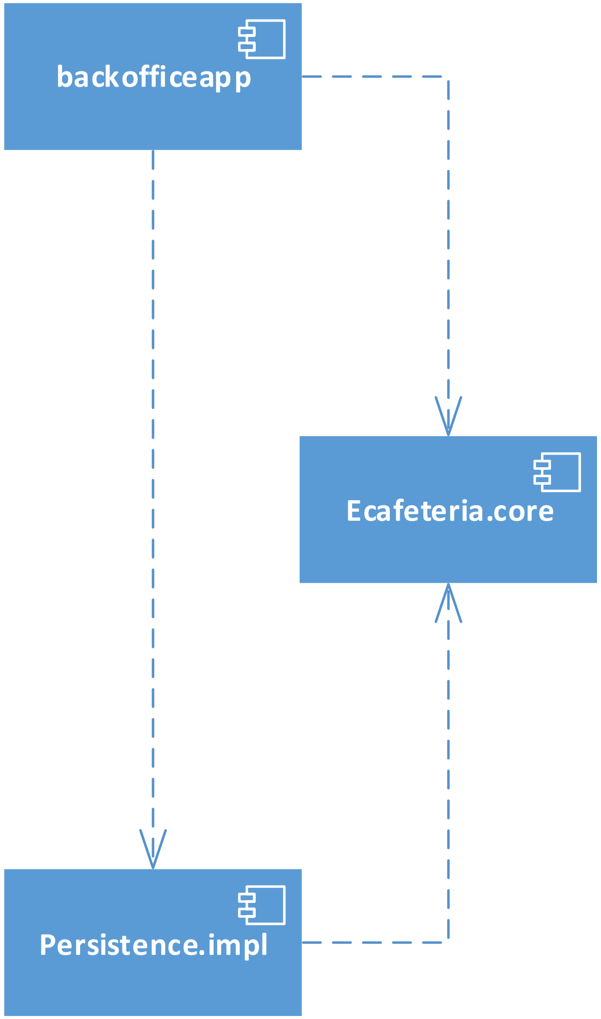


```
# To change this template, choose Tools | Templates
# and open the template in the editor.

persistence.repositoryFactory=eapli.ecafeteria.persistence.jpa.JpaRepositoryFactory
#persistence.repositoryFactory=eapli.ecafeteria.persistence.inmemory.InMemoryReposit
ui.menu.layout=horizontal
persistence.persistenceUnit=eapli.eCafeteriaPU
```

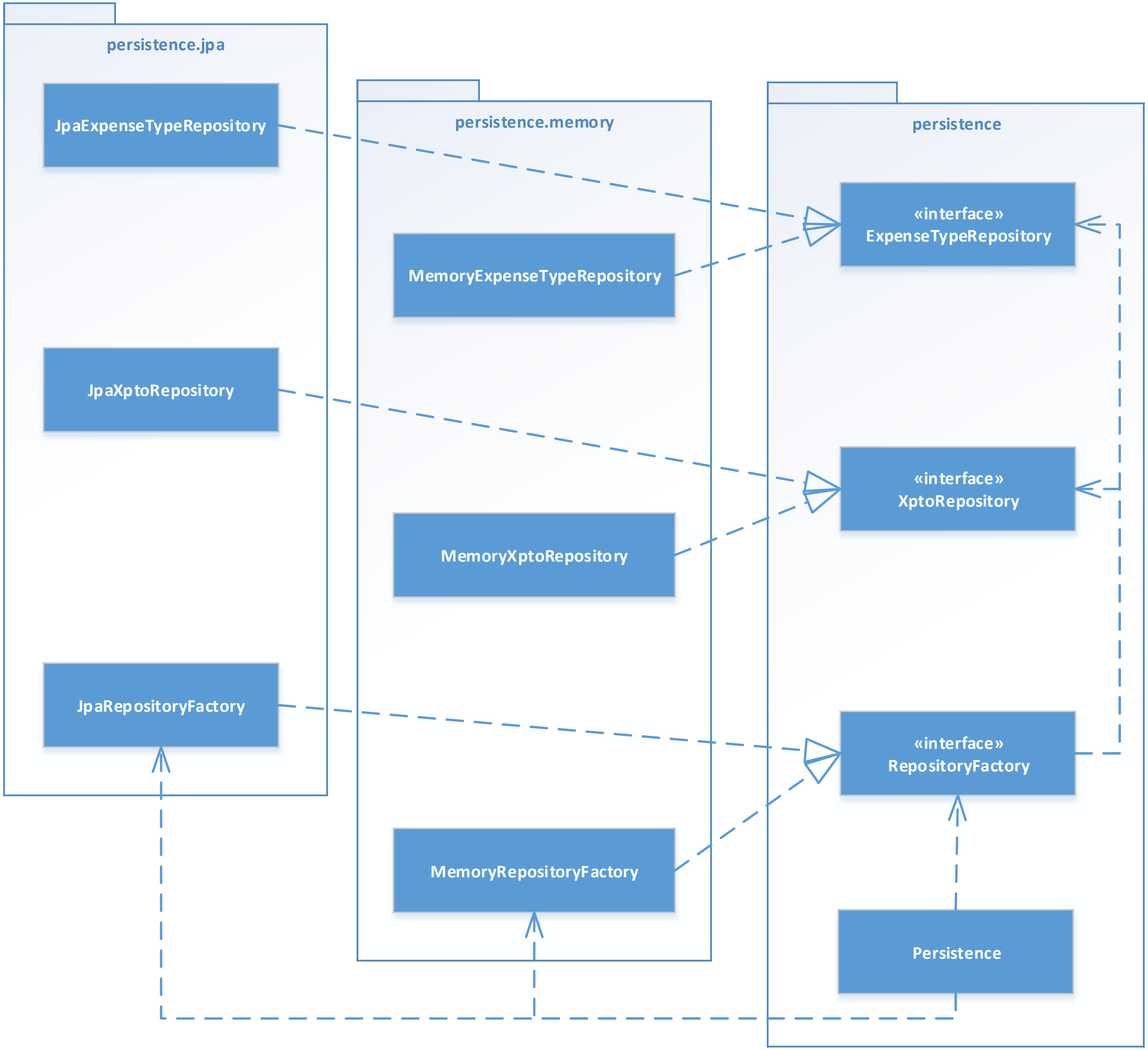
Persistence

- Controller needs to access the repositories
- But we want to have two repository implementations
  - In memory
  - Relational database
- Hide persistence details from rest of the code
  - Interfaces
  - Factories

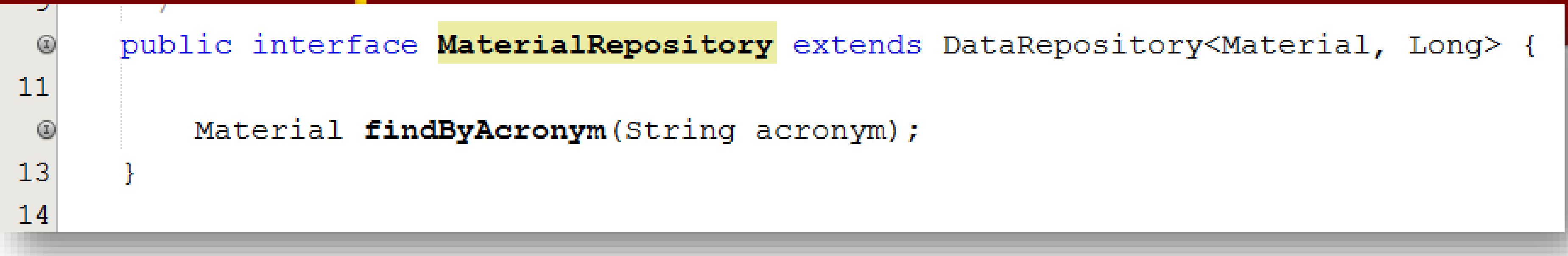
Persistence



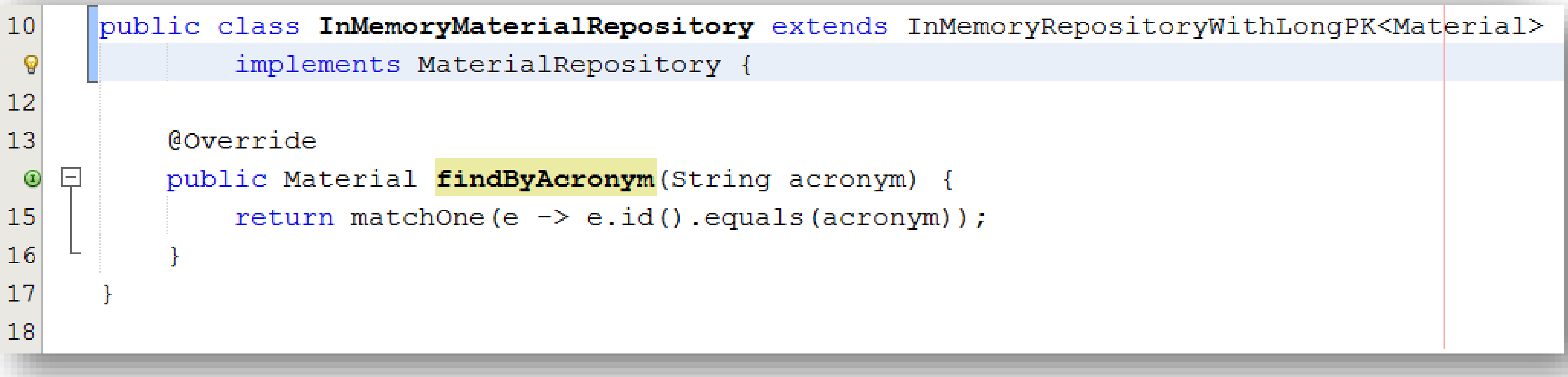
- Separate the definition of repositories (core) from the actual implementation (persistence.impl)
- Apply "Abstract Factory" GoF pattern


persistence.jpa
JpaExpenseTypeRepository
JpaXptoRepository
JpaRepositoryFactory
persistence.memory
MemoryExpenseTypeRepository
MemoryXptoRepository
MemoryRepositoryFactory
persistence
«interface»
ExpenseTypeRepository
«interface»
XptoRepository
«interface»
RepositoryFactory
Persistence

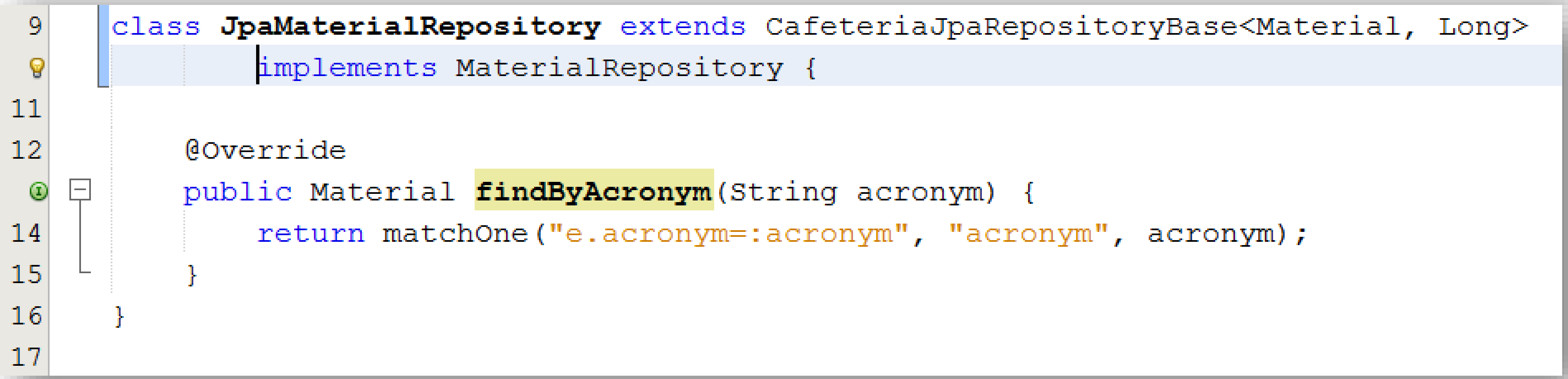
Example



```java
public interface MaterialRepository extends DataRepository<Material, Long> {

    Material findByAcronym(String acronym);
}
```

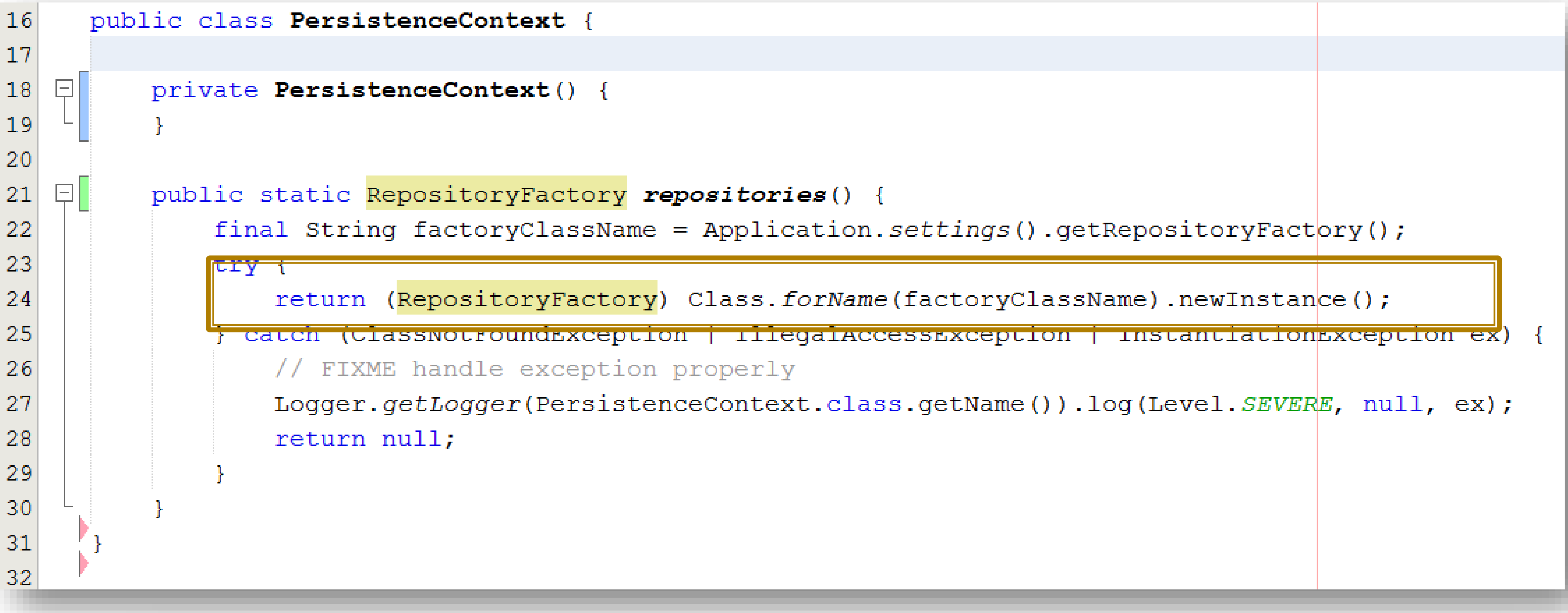


```java
public class InMemoryMaterialRepository extends InMemoryRepositoryWithLongPK<Material>
        implements MaterialRepository {

    @Override
    public Material findByAcronym(String acronym) {
        return matchOne(e -> e.id().equals(acronym));
    }
}
```

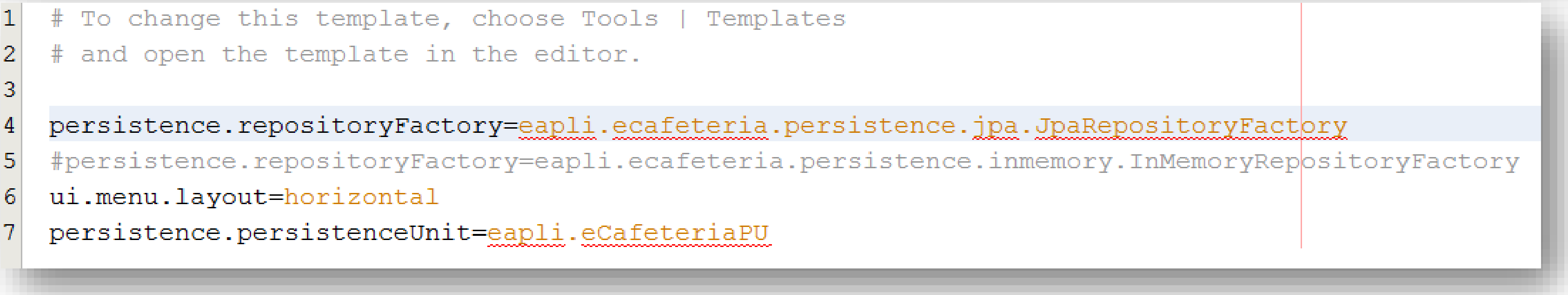


```java
class JpaMaterialRepository extends CafeteriaJpaRepositoryBase<Material, Long>
        implements MaterialRepository {

    @Override
    public Material findByAcronym(String acronym) {
        return matchOne("e.acronym=:acronym", "acronym", acronym);
    }
}
```

Persistence Context

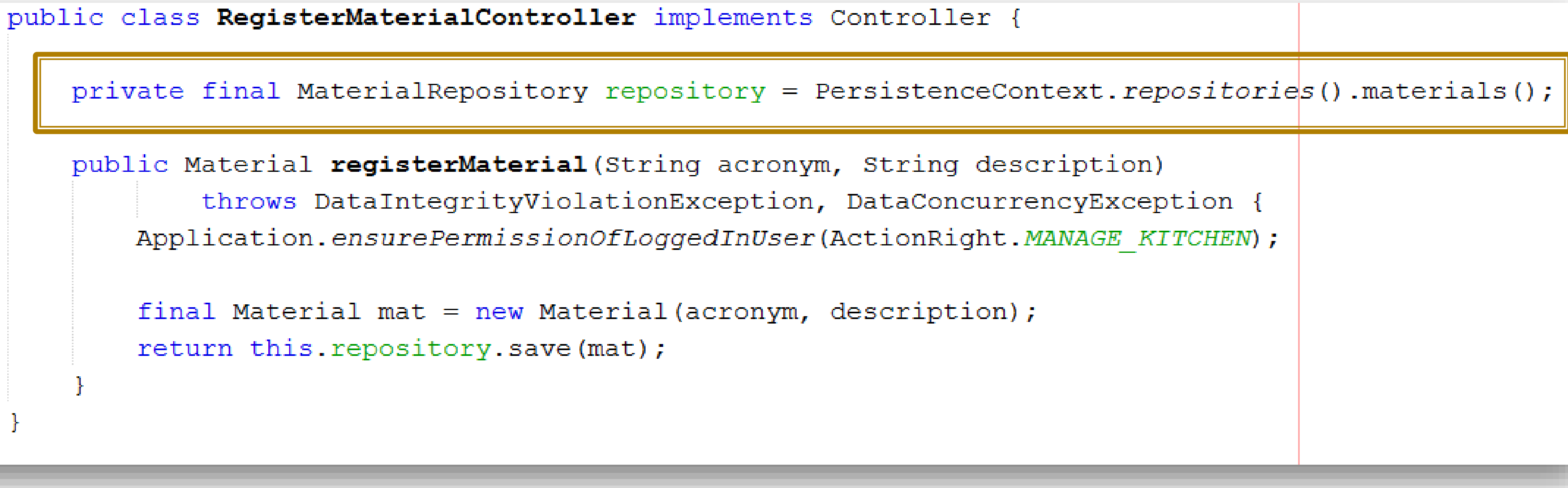


```java
public class PersistenceContext {

    private PersistenceContext() {
    }

    public static RepositoryFactory repositories() {
        final String factoryClassName = Application.settings().getRepositoryFactory();
        try {
            return (RepositoryFactory) Class.forName(factoryClassName).newInstance();
        } catch (ClassNotFoundException | IllegalAccessException | InstantiationException ex) {
            // FIXME handle exception properly
            Logger.getLogger(PersistenceContext.class.getName()).log(Level.SEVERE, null, ex);
            return null;
        }
    }
}
```



```
# To change this template, choose Tools | Templates
# and open the template in the editor.

persistence.repositoryFactory=eapli.ecafeteria.persistence.jpa.JpaRepositoryFactory
#persistence.repositoryFactory=eapli.ecafeteria.persistence.inmemory.InMemoryRepositoryFactory
ui.menu.layout=horizontal
persistence.persistenceUnit=eapli.eCafeteriaPU
```

Persistence Context Usage

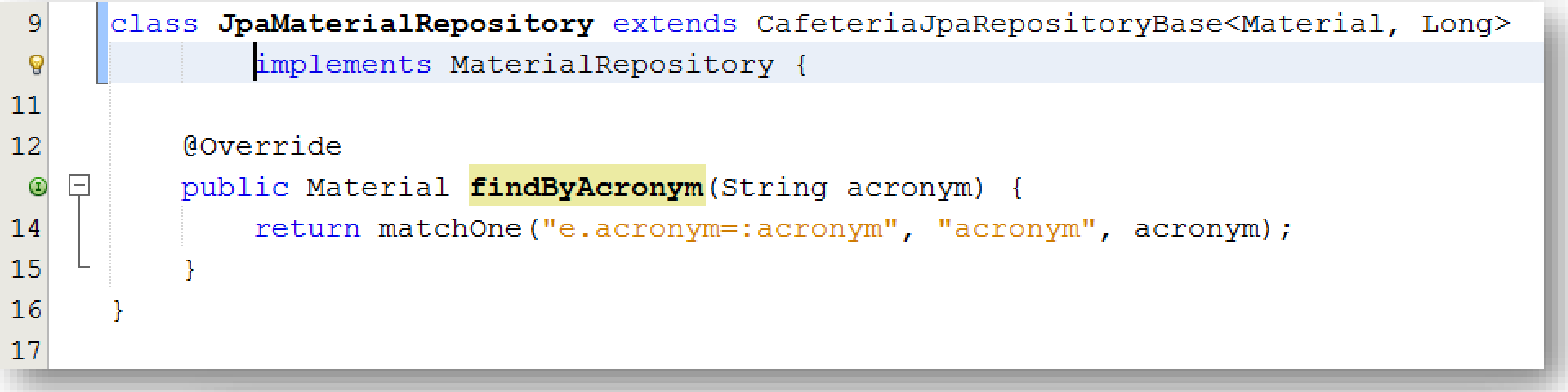


```
public class RegisterMaterialController implements Controller {

    private final MaterialRepository repository = PersistenceContext.repositories().materials();

    public Material registerMaterial(String acronym, String description)
            throws DataIntegrityViolationException, DataConcurrencyException {
        Application.ensurePermissionOfLoggedInUser(ActionRight.MANAGE_KITCHEN);

        final Material mat = new Material(acronym, description);
        return this.repository.save(mat);
    }
}
```

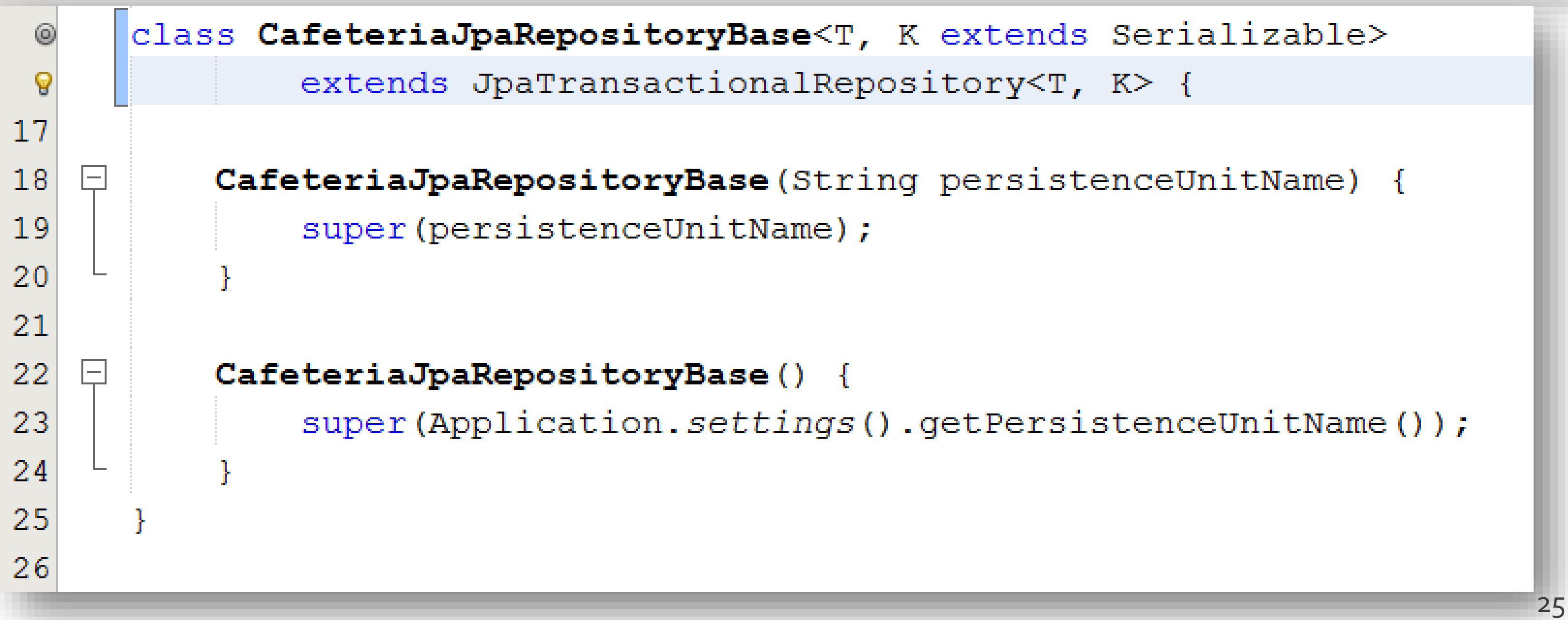
JPA Repositories (framework)

- JpaBaseRepository
  - Generic repository implementation that expects the entity manager factory to be injected by a container, e.g., web server
- JpaNotRunningInContainerBaseRepository
  - For scenarios where the code is not running in a container but transaction is managed by the outside, e.g., controller
- JpaTransactionalBaseRepository
  - For scenarios not running in a container but transactions are created and committed by each repository method; the connection is also closed automatically in each method.
- JpaAutoTxRepository
  - Dual behaviour to either have outside transactional control or explicit transaction in each method

Full transaction control by the repository



```java
class JpaMaterialRepository extends CafeteriaJpaRepositoryBase<Material, Long>
        implements MaterialRepository {

    @Override
    public Material findByAcronym(String acronym) {
        return matchOne("e.acronym=:acronym", "acronym", acronym);
    }
}

```

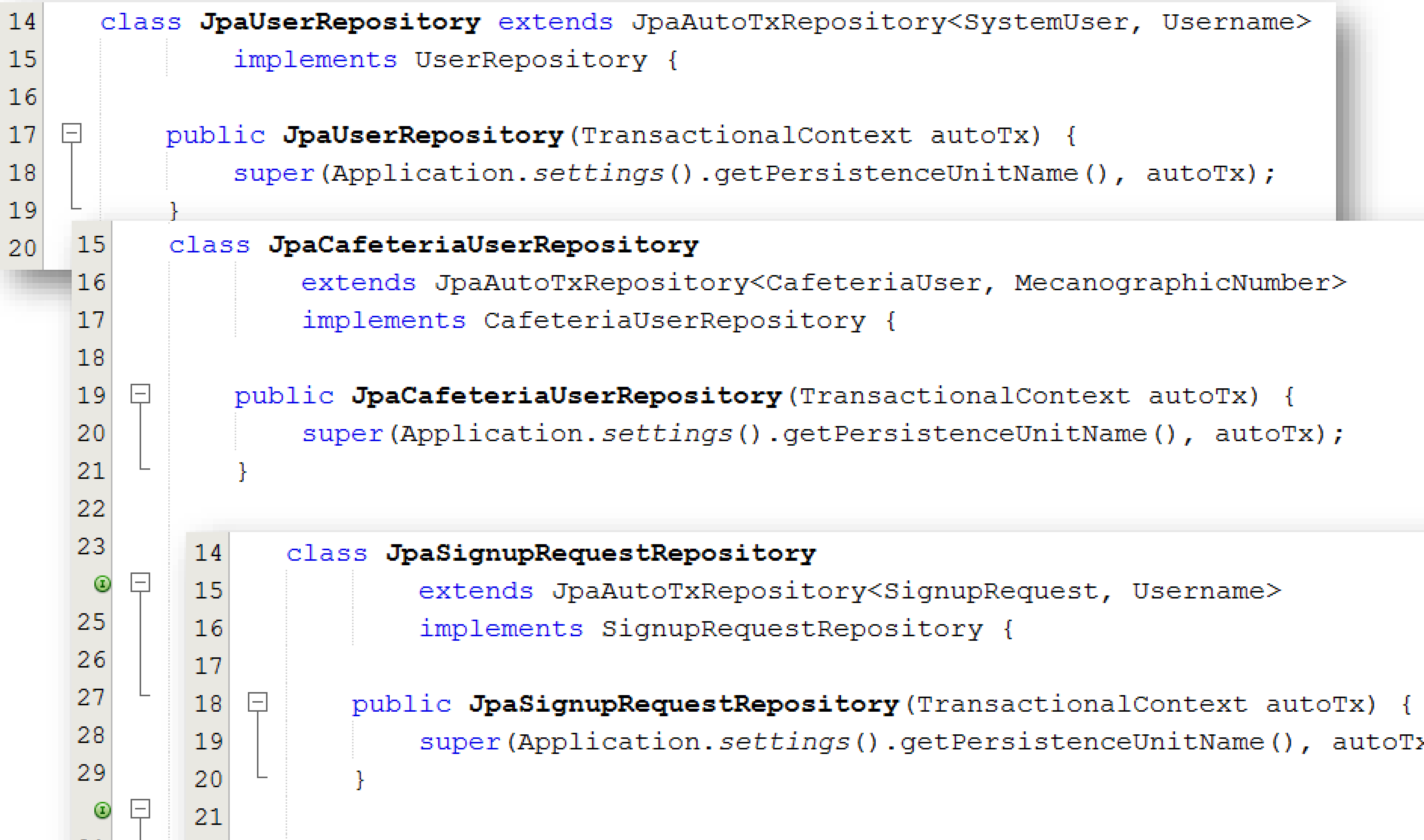


```java
class CafeteriaJpaRepositoryBase<T, K extends Serializable>
        extends JpaTransactionalRepository<T, K> {

    CafeteriaJpaRepositoryBase(String persistenceUnitName) {
        super(persistenceUnitName);
    }

    CafeteriaJpaRepositoryBase() {
        super(Application.settings().getPersistenceUnitName());
    }
}

```

Transaction control (1)

- Accepting a signup request needs to
  - Create a system user
  - Create a cafeteria user
  - Change the status of the signup request
- Three different aggregates!

Transaction control (2): use JpaAutoTxRepository

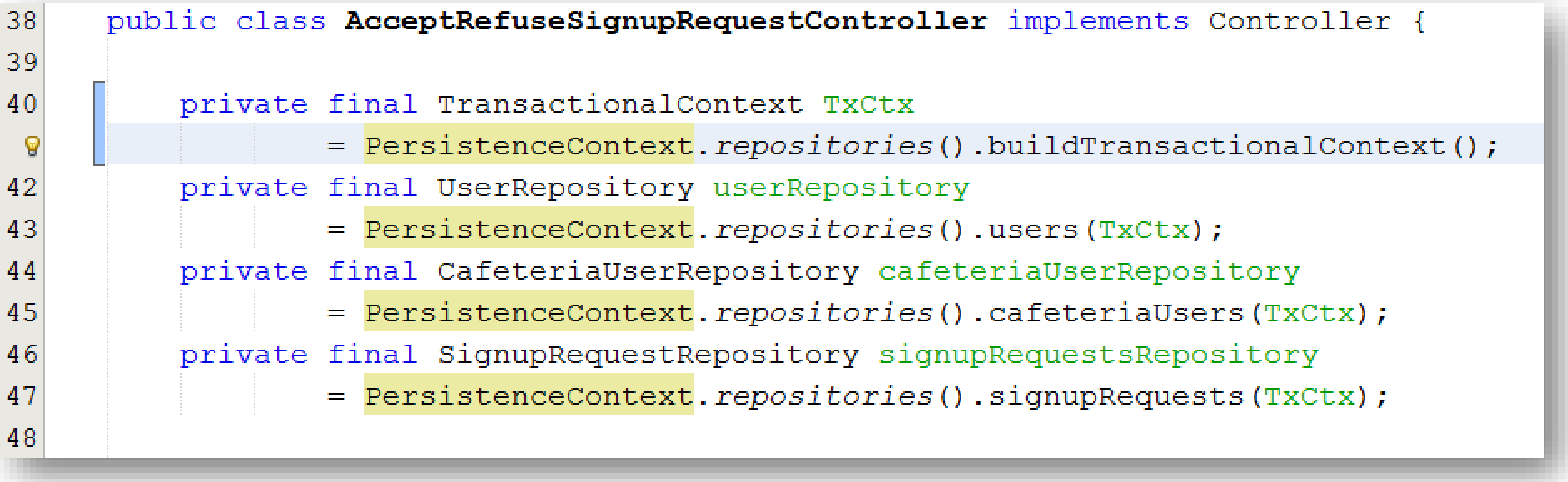


```
class JpaUserRepository extends JpaAutoTxRepository<SystemUser, Username>
        implements UserRepository {

    public JpaUserRepository(TransactionalContext autoTx) {
        super(Application.settings().getPersistenceUnitName(), autoTx);
    }
```

```
class JpaCafeteriaUserRepository
        extends JpaAutoTxRepository<CafeteriaUser, MecanographicNumber>
        implements CafeteriaUserRepository {

    public JpaCafeteriaUserRepository(TransactionalContext autoTx) {
        super(Application.settings().getPersistenceUnitName(), autoTx);
    }
```

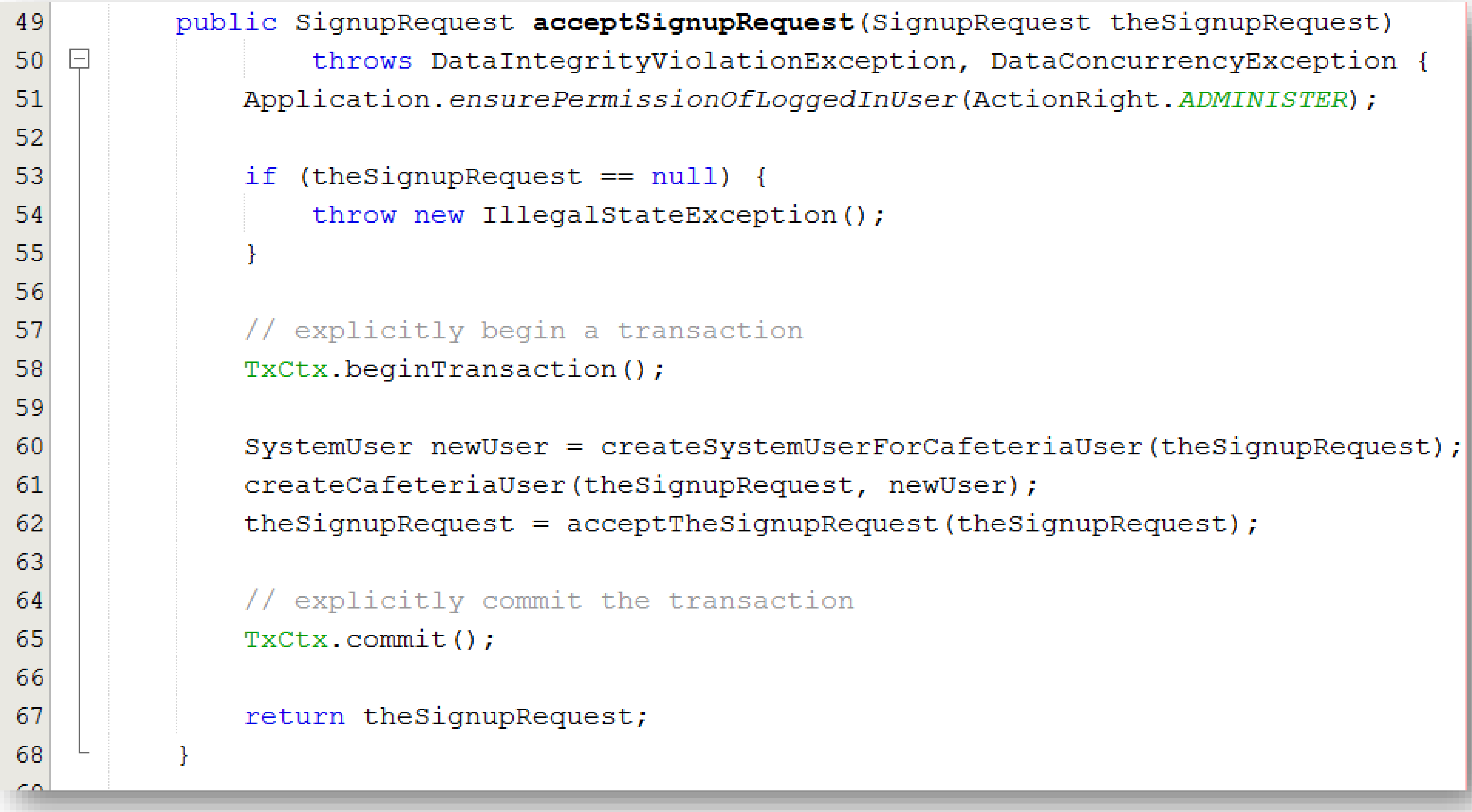
```
class JpaSignupRequestRepository
        extends JpaAutoTxRepository<SignupRequest, Username>
        implements SignupRequestRepository {

    public JpaSignupRequestRepository(TransactionalContext autoTx) {
        super(Application.settings().getPersistenceUnitName(), autoT
    }
```

Transaction control (3): explicit control by the controller



```
public class AcceptRefuseSignupRequestController implements Controller {

    private final TransactionalContext TxCtx
            = PersistenceContext.repositories().buildTransactionalContext();
    private final UserRepository userRepository
            = PersistenceContext.repositories().users(TxCtx);
    private final CafeteriaUserRepository cafeteriaUserRepository
            = PersistenceContext.repositories().cafeteriaUsers(TxCtx);
    private final SignupRequestRepository signupRequestsRepository
            = PersistenceContext.repositories().signupRequests(TxCtx);

```

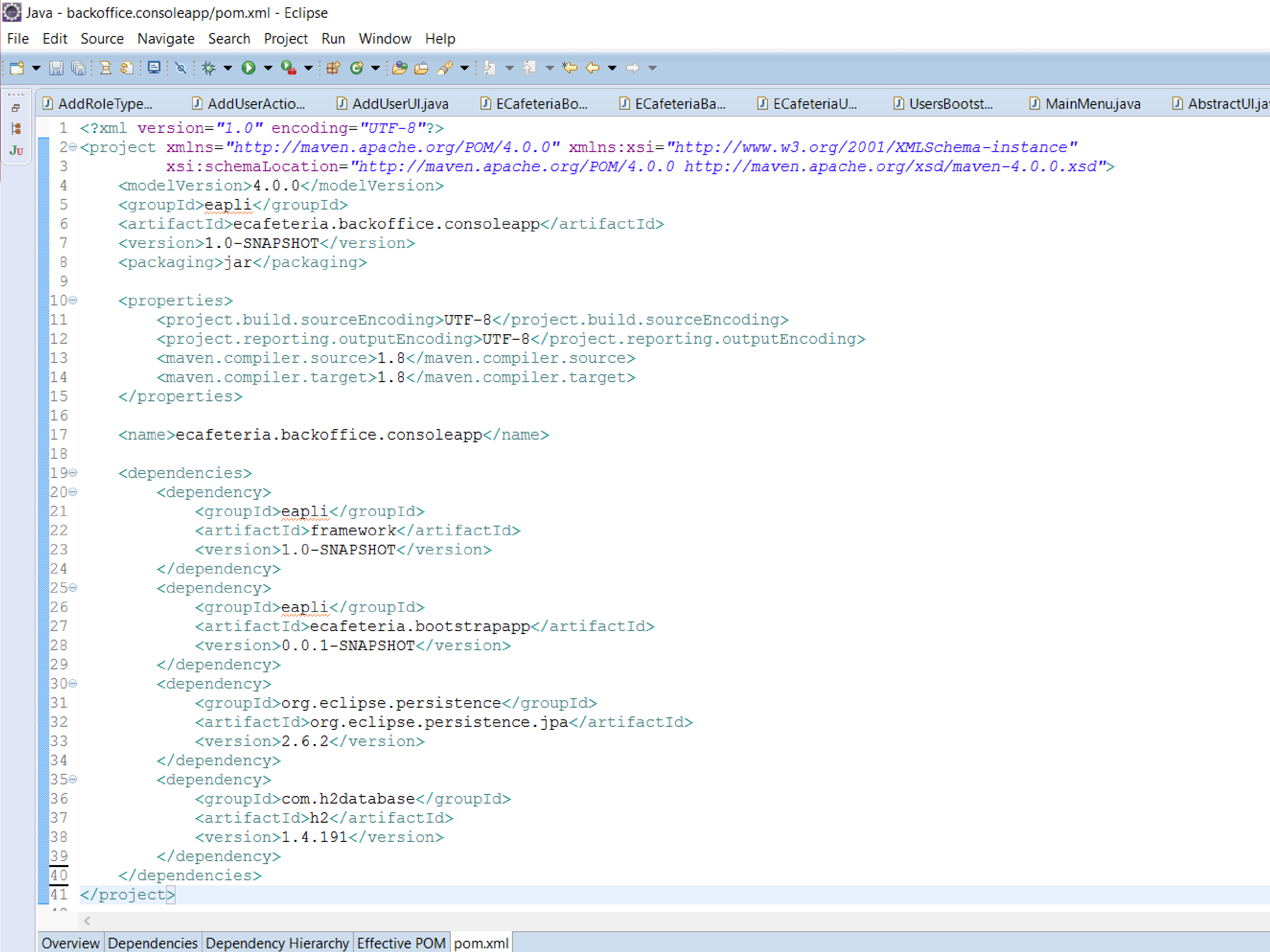
Transaction control (3): explicit control by the controller



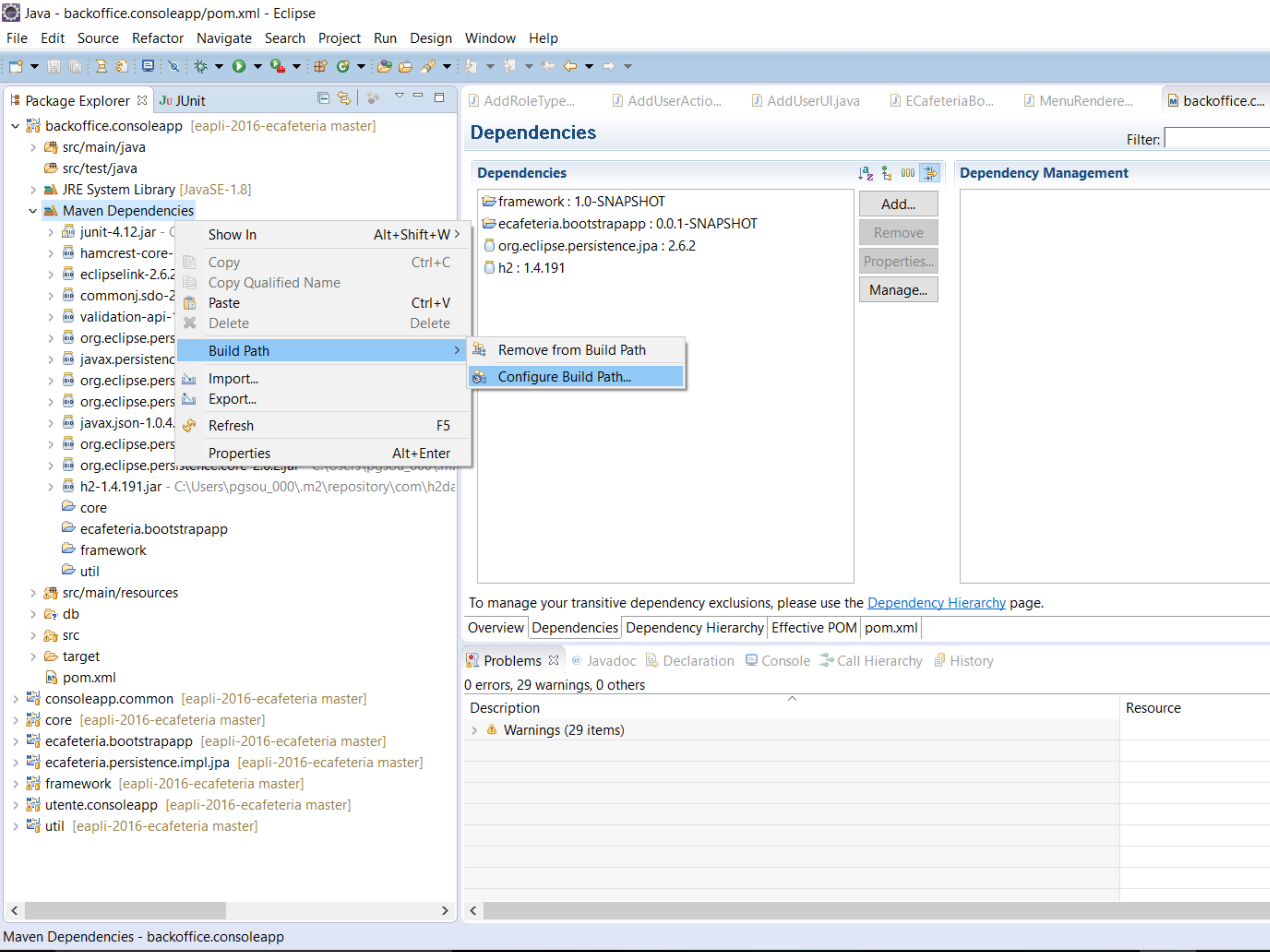
```java
    public SignupRequest acceptSignupRequest(SignupRequest theSignupRequest)
            throws DataIntegrityViolationException, DataConcurrencyException {
        Application.ensurePermissionOfLoggedInUser(ActionRight.ADMINISTER);

        if (theSignupRequest == null) {
            throw new IllegalStateException();
        }

        // explicitly begin a transaction
        TxCtx.beginTransaction();

        SystemUser newUser = createSystemUserForCafeteriaUser(theSignupRequest);
        createCafeteriaUser(theSignupRequest, newUser);
        theSignupRequest = acceptTheSignupRequest(theSignupRequest);

        // explicitly commit the transaction
        TxCtx.commit();

        return theSignupRequest;
    }
```

Build

- Maven
  - Dependency manager
  - Artifact (jar) repository
  - Build automation
  - Other tasks, e.g., deploy, run
- Works for Eclipse, InteliJ, Netbeans
- Pom.xml



```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>eapli</groupId>
    <artifactId>ecafeteria.backoffice.consoleapp</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <name>ecafeteria.backoffice.consoleapp</name>

    <dependencies>
        <dependency>
            <groupId>eapli</groupId>
            <artifactId>framework</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>eapli</groupId>
            <artifactId>ecafeteria.bootstrapapp</artifactId>
            <version>0.0.1-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>org.eclipse.persistence</groupId>
            <artifactId>org.eclipse.persistence.jpa</artifactId>
            <version>2.6.2</version>
        </dependency>
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <version>1.4.191</version>
        </dependency>
    </dependencies>
</project>
```



To manage your transitive dependency exclusions, please use the Dependency Hierarchy page.

0 errors, 29 warnings, 0 others

Maven Dependencies - backoffice.consoleapp

Use cases implemented in base project

- Add user
- List users
- Deactivate user
- Check permissions
- Signup
- Approve new user
- Add dish type
- Edit dish type
- Deactivate dish type
- List dish types
- Register organic unit
- Add dish
- Add Material

Next steps

1. Read project description
2. Discuss and clear assumptions in PL
3. Clone class' repository
   - One for each PL class
4. Study base code
5. Analyse – design – code – test – document

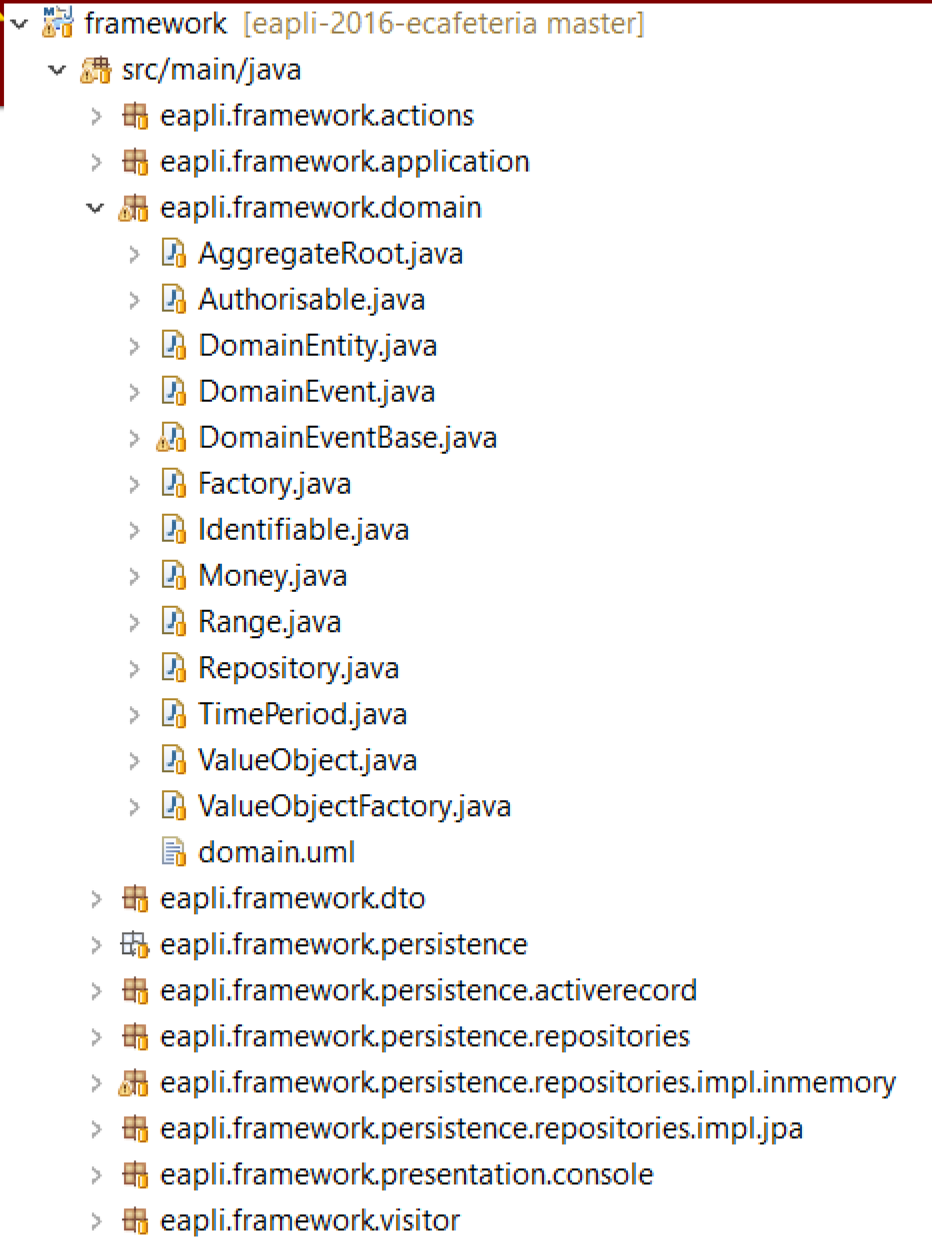
EAPLI Framework

Eapli.Util

- Console
  - Helper console reading functions
- DateTime
  - Simplifies manipulation of dates and times thru java Calendar
- Files
  - File manipulation helper
- Math
  - Sample math utility
- RomanNumeral
  - Represents a decimal number as a Roman numeral
- Strings
  - String manipulation

Eapli.Framework

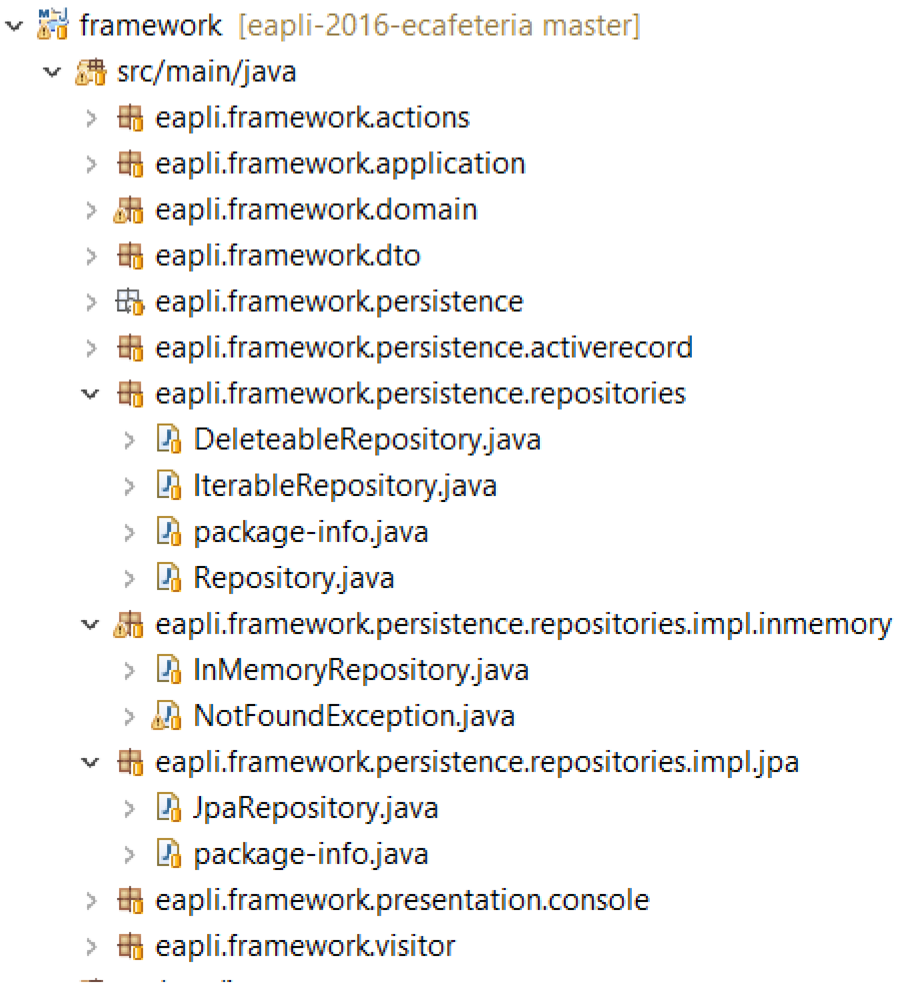
- Domain objects
  - Money
  - Range
  - Time period
- DDD pattern interfaces
  - ValueObject
  - DomainEntity
  - AggregateRoot



```
framework [eapli-2016-ecafeteria master]
  src/main/java
    eapli.framework.actions
    eapli.framework.application
    eapli.framework.domain
      AggregateRoot.java
      Authorisable.java
      DomainEntity.java
      DomainEvent.java
      DomainEventBase.java
      Factory.java
      Identifiable.java
      Money.java
      Range.java
      Repository.java
      TimePeriod.java
      ValueObject.java
      ValueObjectFactory.java
      domain.uml
    eapli.framework.dto
    eapli.framework.persistence
    eapli.framework.persistence.activerecord
    eapli.framework.persistence.repositories
    eapli.framework.persistence.repositories.impl.inmemory
    eapli.framework.persistence.repositories.impl.jpa
    eapli.framework.presentation.console
    eapli.framework.visitor
```

Eapli.Framework

- Persistence
  - Repository interfaces
- Implementations
  - JPA
  - InMemory list



```
framework  [eapli-2016-ecafeteria master]
  src/main/java
    eapli.framework.actions
    eapli.framework.application
    eapli.framework.domain
    eapli.framework.dto
    eapli.framework.persistence
    eapli.framework.persistence.activerecord
    eapli.framework.persistence.repositories
      DeleteableRepository.java
      IterableRepository.java
      package-info.java
      Repository.java
    eapli.framework.persistence.repositories.impl.inmemory
      InMemoryRepository.java
      NotFoundException.java
    eapli.framework.persistence.repositories.impl.jpa
      JpaRepository.java
      package-info.java
    eapli.framework.presentation.console
    eapli.framework.visitor
```

Repositories

- DataRepository
- TransactionalContext

- Interfaces for describing repository functionalities and transactions

In memory Repositories

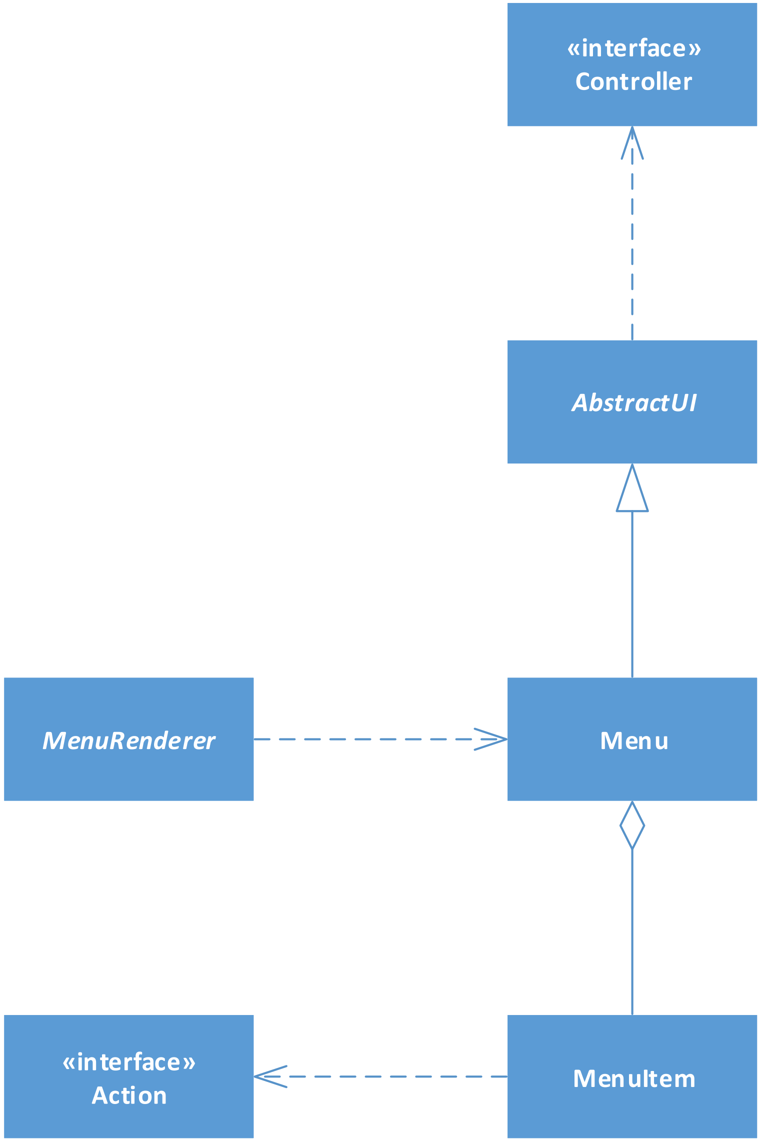
- InMemoryBaseRepository
- InMemoryBaseRepositorywithLongPK

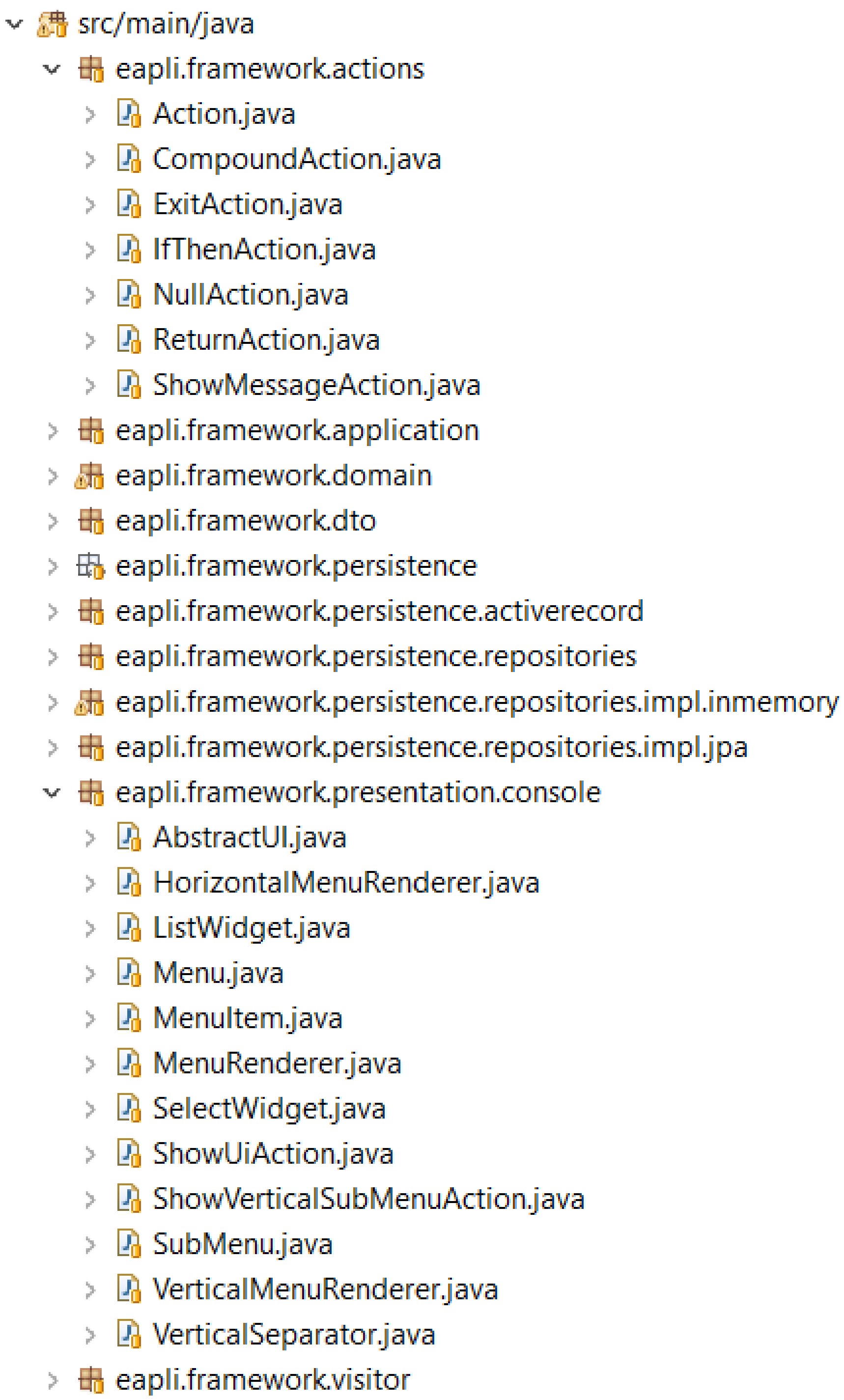
- For pedagogical testing purposes only!

JPA Repositories

- JpaBaseRepository
  - Generic repository implementation that expects the entity manager factory to be injected by a container, e.g., web server
- JpaNotRunningInContainerBaseRepository
  - For scenarios where the code is not running in a container but transaction is managed by the outside, e.g., controller
- JpaTransactionalBaseRepository
  - For scenarios not running in a container but transactions are created and committed by each repository method; the connection is also closed automatically in each method.
- JpaAutoTxRepository
  - Dual behaviour to either have outside transactional control or explicit transaction in each method

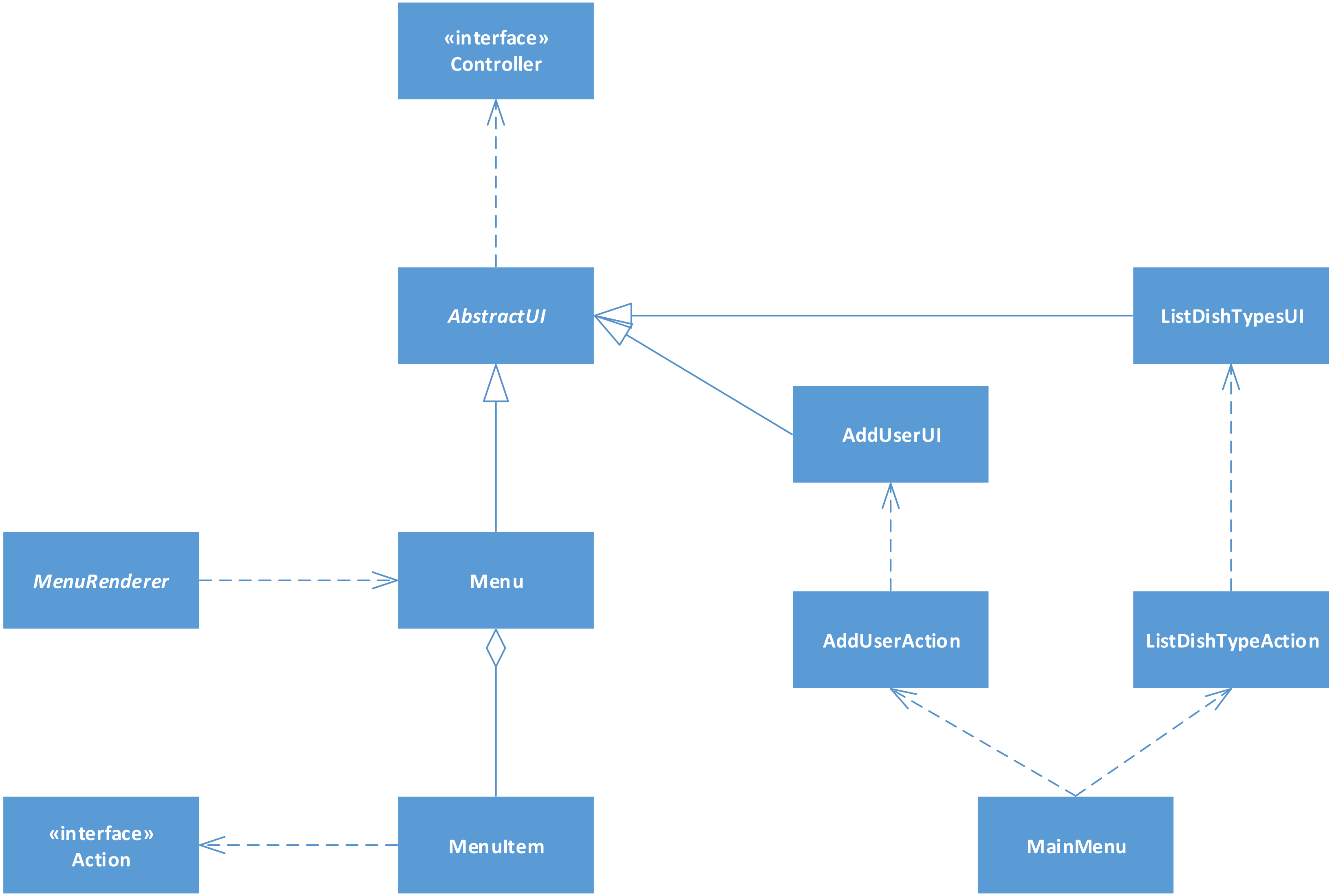
Presentation

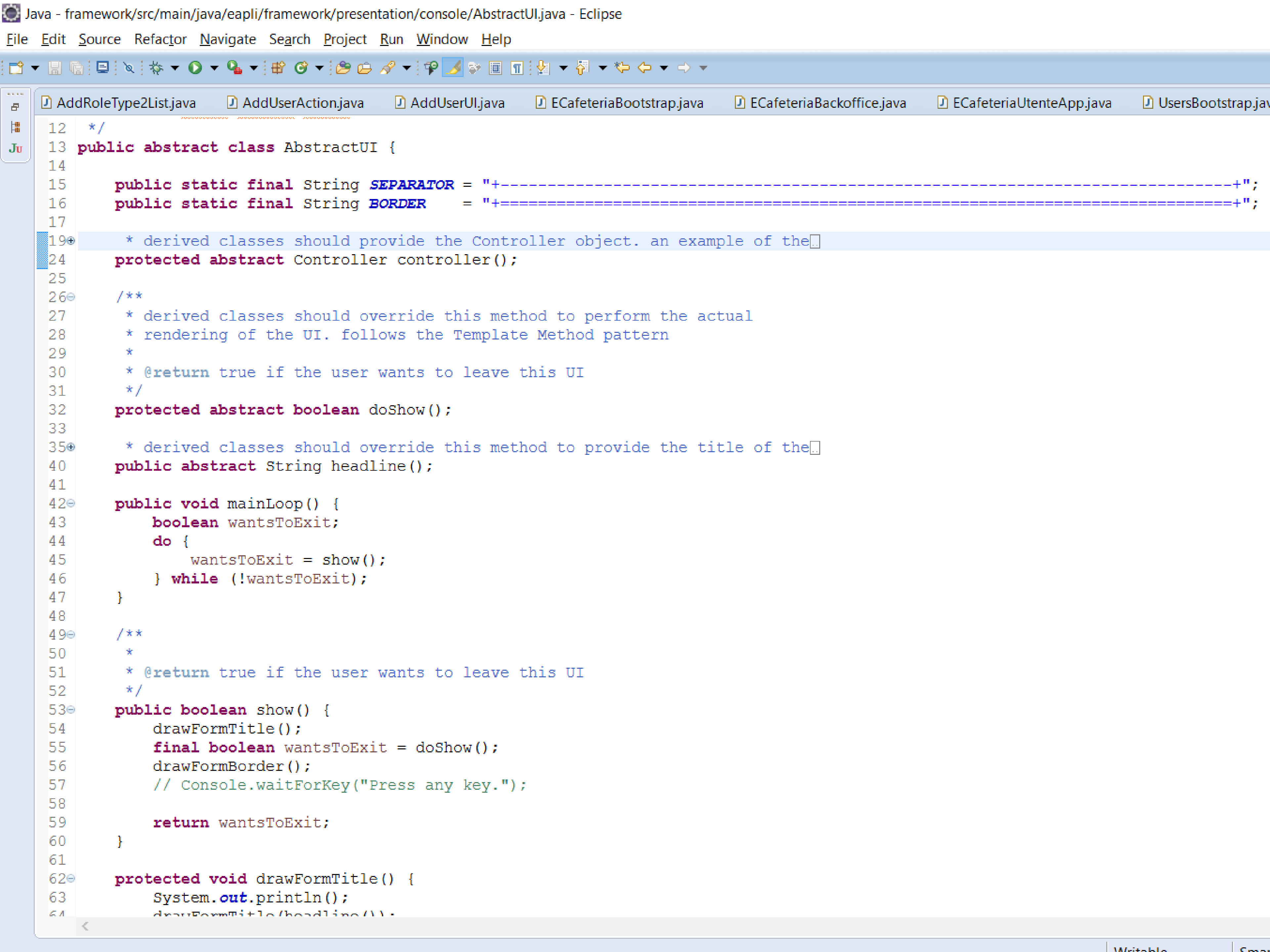




```
src/main/java
  eapli.framework.actions
    Action.java
    CompoundAction.java
    ExitAction.java
    IfThenAction.java
    NullAction.java
    ReturnAction.java
    ShowMessageAction.java
  eapli.framework.application
  eapli.framework.domain
  eapli.framework.dto
  eapli.framework.persistence
  eapli.framework.persistence.activerecord
  eapli.framework.persistence.repositories
  eapli.framework.persistence.repositories.impl.inmemory
  eapli.framework.persistence.repositories.impl.jpa
  eapli.framework.presentation.console
    AbstractUI.java
    HorizontalMenuRenderer.java
    ListWidget.java
    Menu.java
    MenuItem.java
    MenuRenderer.java
    SelectWidget.java
    ShowUiAction.java
    ShowVerticalSubMenuAction.java
    SubMenu.java
    VerticalMenuRenderer.java
    VerticalSeparator.java
  eapli.framework.visitor
```

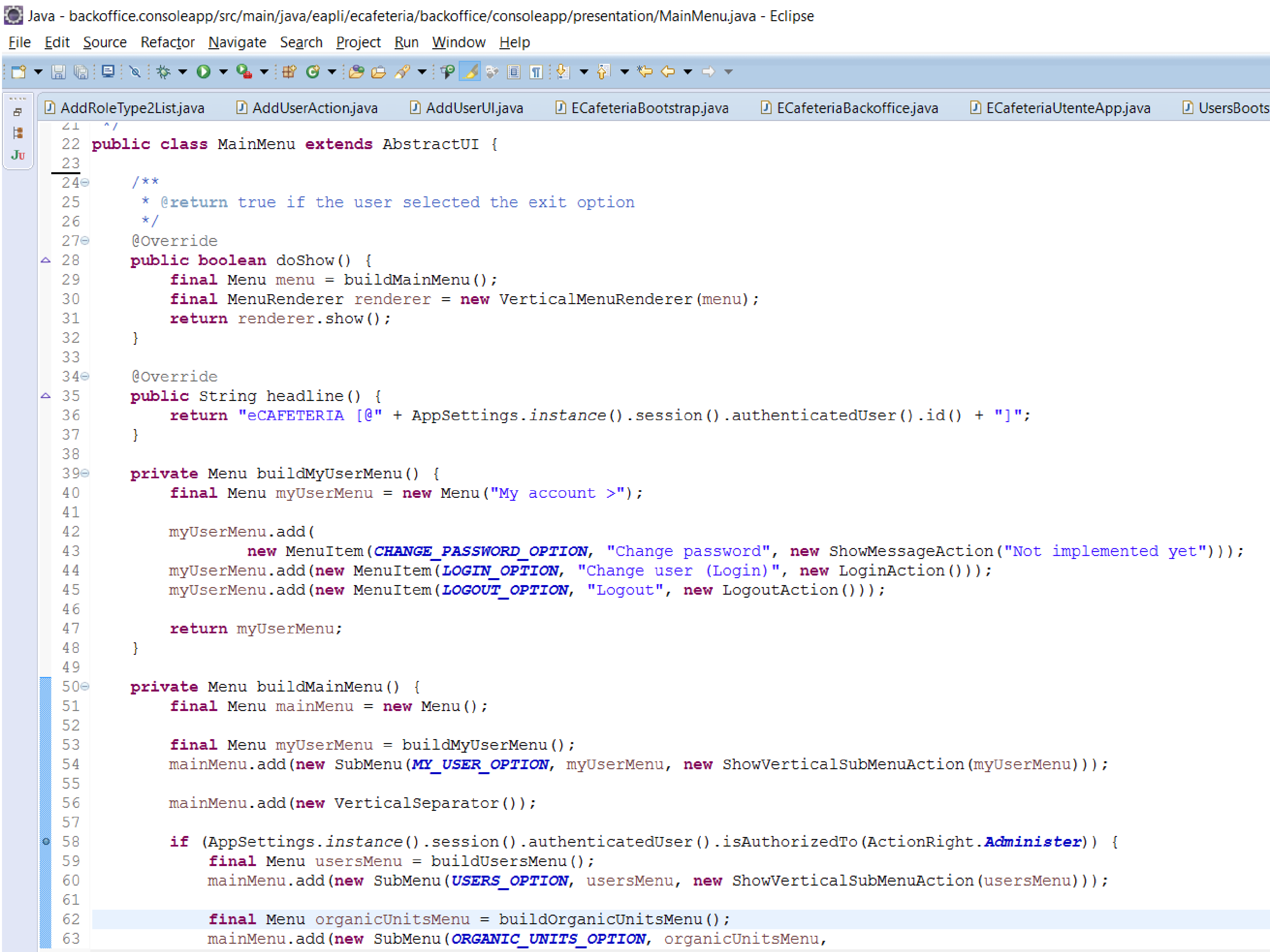
presentation





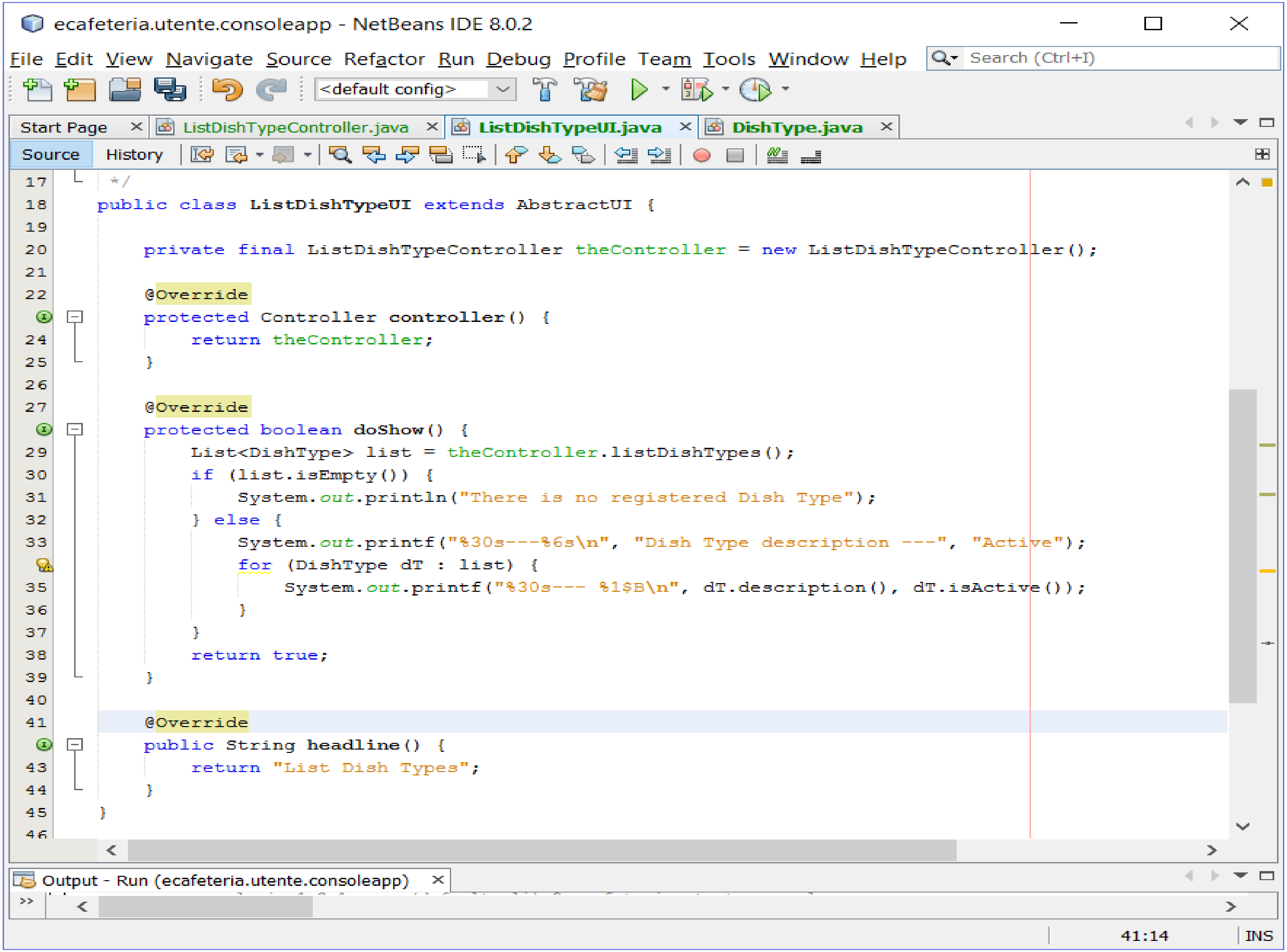
Java - framework/src/main/java/eapli/framework/presentation/console/AbstractUI.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
AddRoleType2List.java
AddUserAction.java
AddUserUI.java
ECafeteriaBootstrap.java
ECafeteriaBackoffice.java
ECafeteriaUtenteApp.java
UsersBootstrap.jav
Writable
Sma

```java
 */
public abstract class AbstractUI {

    public static final String SEPARATOR = "+--------------------------------------------------------------------------------+";
    public static final String BORDER    = "+================================================================================+";

     * derived classes should provide the Controller object. an example of the
    protected abstract Controller controller();

    /**
     * derived classes should override this method to perform the actual
     * rendering of the UI. follows the Template Method pattern
     *
     * @return true if the user wants to leave this UI
     */
    protected abstract boolean doShow();

     * derived classes should override this method to provide the title of the
    public abstract String headline();

    public void mainLoop() {
        boolean wantsToExit;
        do {
            wantsToExit = show();
        } while (!wantsToExit);
    }

    /**
     *
     * @return true if the user wants to leave this UI
     */
    public boolean show() {
        drawFormTitle();
        final boolean wantsToExit = doShow();
        drawFormBorder();
        // Console.waitForKey("Press any key.");

        return wantsToExit;
    }

    protected void drawFormTitle() {
        System.out.println();
        drawFormTitle(headline());
```



Java - backoffice.consoleapp/src/main/java/eapli/ecafeteria/backoffice/consoleapp/presentation/MainMenu.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
AddRoleType2List.java
AddUserAction.java
AddUserUI.java
ECafeteriaBootstrap.java
ECafeteriaBackoffice.java
ECafeteriaUtenteApp.java
UsersBoots

```
 */
public class MainMenu extends AbstractUI {

    /**
     * @return true if the user selected the exit option
     */
    @Override
    public boolean doShow() {
        final Menu menu = buildMainMenu();
        final MenuRenderer renderer = new VerticalMenuRenderer(menu);
        return renderer.show();
    }

    @Override
    public String headline() {
        return "eCAFETERIA [@" + AppSettings.instance().session().authenticatedUser().id() + "]";
    }

    private Menu buildMyUserMenu() {
        final Menu myUserMenu = new Menu("My account >");

        myUserMenu.add(
                new MenuItem(CHANGE_PASSWORD_OPTION, "Change password", new ShowMessageAction("Not implemented yet")));
        myUserMenu.add(new MenuItem(LOGIN_OPTION, "Change user (Login)", new LoginAction()));
        myUserMenu.add(new MenuItem(LOGOUT_OPTION, "Logout", new LogoutAction()));

        return myUserMenu;
    }

    private Menu buildMainMenu() {
        final Menu mainMenu = new Menu();

        final Menu myUserMenu = buildMyUserMenu();
        mainMenu.add(new SubMenu(MY_USER_OPTION, myUserMenu, new ShowVerticalSubMenuAction(myUserMenu)));

        mainMenu.add(new VerticalSeparator());

        if (AppSettings.instance().session().authenticatedUser().isAuthorizedTo(ActionRight.Administer)) {
            final Menu usersMenu = buildUsersMenu();
            mainMenu.add(new SubMenu(USERS_OPTION, usersMenu, new ShowVerticalSubMenuAction(usersMenu)));

            final Menu organicUnitsMenu = buildOrganicUnitsMenu();
            mainMenu.add(new SubMenu(ORGANIC_UNITS_OPTION, organicUnitsMenu,
```


ecafeteria.utente.consoleapp - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
Start Page
ListDishTypeController.java
ListDishTypeUI.java
DishType.java
Source
History
Output - Run (ecafeteria.utente.consoleapp)
41:14
INS

```
 */
public class ListDishTypeUI extends AbstractUI {

    private final ListDishTypeController theController = new ListDishTypeController();

    @Override
    protected Controller controller() {
        return theController;
    }

    @Override
    protected boolean doShow() {
        List<DishType> list = theController.listDishTypes();
        if (list.isEmpty()) {
            System.out.println("There is no registered Dish Type");
        } else {
            System.out.printf("%30s---%6s\n", "Dish Type description ---", "Active");
            for (DishType dT : list) {
                System.out.printf("%30s--- %1$B\n", dT.description(), dT.isActive());
            }
        }
        return true;
    }

    @Override
    public String headline() {
        return "List Dish Types";
    }
}
```