Kenzan challenge code

Design patterns

Active record pattern

For the Data base I used the active record pattern, an active record is an object, which:

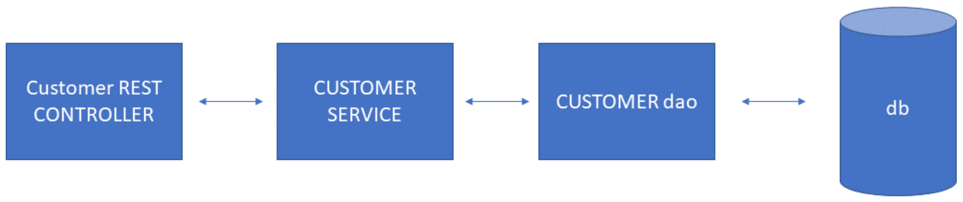
represents an object in the domain because I followed the business rules, knows how to handle certain operations on the object for the manipulation.

knows how to retrieve, update, save and delete the entity.

Employee

`first_name`
`middle_initial`
`last_name`
`lastName;`
`date_of_birth`
`date_of_employment`
`email`
`status`

Insert Update getException



Factory Method

I used this design pattern to the object creation, I define a class whish is belong to a logic segment, implementing a interface. Then I created a factory for these employee object, whit the help of hibernate.

```
public interface EmployeeDAO {

	public List<Employee> getEmployees();

	public void saveEmployee(Employee theEmployee);

	public Employee getEmployee(int theId);

	public void deleteEmployee(int theId);

}
```

Then I define a class that implement this interface:

```
public class EmployeeDAOImpl implements EmployeeDAO {

	// need to inject the session factory
	@Autowired
	private SessionFactory sessionFactory;

	@Override
	public List<Employee> getEmployees() {

		// get the current hibernate session
		Session currentSession = 
sessionFactory.getCurrentSession();

		// create a query  ... sort by last name
		Query<Employee> theQuery = 
				currentSession.createQuery("from Employee 
order by lastName",
				
	Employee.class);

		// execute query and get result list
		List<Employee> employees = theQuery.getResultList();

		// return the results
		return employees;
	}

	@Override
	public void saveEmployee(Employee theEmployee) {

		// get current hibernate session
		Session currentSession = 
sessionFactory.getCurrentSession();

		// save/upate the employee ... finally LOL
		currentSession.saveOrUpdate(theEmployee);

	}

	@Override
	public Employee getEmployee(int theId) {

		// get the current hibernate session
		Session currentSession = 
sessionFactory.getCurrentSession();

		// now retrieve/read from database using the primary key
		Employee theEmployee = 
currentSession.get(Employee.class, theId);
```

```
		return theEmployee;
	}

	@Override
	public void deleteEmployee(int theId) {

		// get the current hibernate session
		Session currentSession = 
sessionFactory.getCurrentSession();

		// delete object with primary key
		Query theQuery = 
					currentSession.createQuery("delete from 
Employee where id=:employeeId");
		theQuery.setParameter("employeeId", theId);

		theQuery.executeUpdate();
	}

}
```

Builder

I used builder design pattern to help build a final object, for my class, this is for the fields and parameters in a step-by-step manner.

```
@Entity
@Table(name="employee")
public class Employee {

	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	@Column(name="id")
	private int id;

	@Column(name="first_name")
	private String firstName;

	@Column(name="middle_initial")
	private String middleInitial;

	@Column(name="last_name")
```

```
private String lastName;

@Column(name="date_of_birth")
private String dateOfBirth;

@Column(name="date_of_employment")
private String dateOfEmployment;

@Column(name="email")
private String email;

@Column(name="status")
private String status;

public Employee() {

}

public int getId() {
	return id;
}

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

public String getFirstName() {
	return firstName;
}

public void setFirstName(String firstName) {
	this.firstName = firstName;
}

public String getLastName() {
	return lastName;
}

public void setLastName(String lastName) {
	this.lastName = lastName;
}

public String getEmail() {
	return email;
}
```

```
	public void setEmail(String email) {
		this.email = email;
	}



	public String getMiddleInitial() {
		return middleInitial;
	}

	public void setMiddleInitial(String middleInitial) {
		this.middleInitial = middleInitial;
	}

	public String getDateOfBirth() {
		return dateOfBirth;
	}

	public void setDateOfBirth(String dateOfBirth) {
		this.dateOfBirth = dateOfBirth;
	}

	public String getDateOfEmployment() {
		return dateOfEmployment;
	}

	public void setDateOfEmployment(String dateOfEmployment) {
		this.dateOfEmployment = dateOfEmployment;
	}

	public String getStatus() {
		return status;
	}

	public void setStatus(String status) {
		this.status = status;
	}


	@Override
	public String toString() {
		return "Employee [id=" + id + ", firstName=" + firstName
+ ", middleInitial=" + middleInitial + ", lastName="
				+ lastName + ", dateOfBirth=" + dateOfBirth +
", dateOfEmployment=" + dateOfEmployment + ", email="
				+ email + ", status=" + status + "]";
	}

}
```

Facade

The facade pattern provides a simple and top-level interface for the client and allows it to access the system, without knowing any of the system logic.

```
public interface EmployeeDAO {

	public List<Employee> getEmployees();

	public void saveEmployee(Employee theEmployee);

	public Employee getEmployee(int theId);

	public void deleteEmployee(int theId);

}
```

This is as much for the creation of my used object and as well as the services because the concrete class that is implement it is below:

```
@Entity
@Table(name="employee")
public class Employee {

	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	@Column(name="id")
	private int id;

	@Column(name="first_name")
	private String firstName;

	@Column(name="middle_initial")
	private String middleInitial;

	@Column(name="last_name")
	private String lastName;

	@Column(name="date_of_birth")
```

```java
private String dateOfBirth;

@Column(name="date_of_employment")
private String dateOfEmployment;

@Column(name="email")
private String email;

@Column(name="status")
private String status;

public Employee() {

}

public int getId() {
	return id;
}

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

public String getFirstName() {
	return firstName;
}

public void setFirstName(String firstName) {
	this.firstName = firstName;
}

public String getLastName() {
	return lastName;
}

public void setLastName(String lastName) {
	this.lastName = lastName;
}

public String getEmail() {
	return email;
}

public void setEmail(String email) {
	this.email = email;
}
```

```
	public String getMiddleInitial() {
		return middleInitial;
	}

	public void setMiddleInitial(String middleInitial) {
		this.middleInitial = middleInitial;
	}

	public String getDateOfBirth() {
		return dateOfBirth;
	}

	public void setDateOfBirth(String dateOfBirth) {
		this.dateOfBirth = dateOfBirth;
	}

	public String getDateOfEmployment() {
		return dateOfEmployment;
	}

	public void setDateOfEmployment(String dateOfEmployment) {
		this.dateOfEmployment = dateOfEmployment;
	}

	public String getStatus() {
		return status;
	}

	public void setStatus(String status) {
		this.status = status;
	}


	@Override
	public String toString() {
		return "Employee [id=" + id + ", firstName=" + firstName
+ ", middleInitial=" + middleInitial + ", lastName="
				+ lastName + ", dateOfBirth=" + dateOfBirth +
", dateOfEmployment=" + dateOfEmployment + ", email="
				+ email + ", status=" + status + "]";
	}

}
```

By using this interface, the users doesn't concern themselves with the logic behind to manipulate and employee.