

به نام خداوند بخشنده مهربان

محمد رضا مسیب زاده

تمرین جاوا :

https://github.com/M-Mosaiebzadeh/Maktab_W17.git

تمرین شماره ۱:

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "footballGame". It contains a "src" directory with "main" and "test" packages. "main" contains "java" and "dao" sub-packages. "java" contains classes like AbstractDao, CityDao, CoachDao, CompetitionDao, ContractDao, DataAccess, MatchEventDao, PlayerDao, StadiumDao, TeamDao, and UserDao. "dao" contains interfaces like ICity, ICoach, ICompetition, IContract, IDataAccess, IMatchEvent, IPlayer, IStadium, ITeam, and IUser. "entities" contains "enums" with MatchEventType (INJURY, YELLOW_CARD, RED_CARD) and entities like City, Coach, Competition, Contract, IEntity, MatchEvent, Player, Stadium, Team, and User. "util" contains EntityManagerFactory.
- Code Editor:** The file "App.java" is open. It imports com.github.javafaker.Faker, dao.* from the dao package, and EntityManager, Date, HashSet, Set, and MatchEventType from the entities package. It also imports static util.EntityManagerFactoryUtil.createEntityManagerFactory and static util.EntityManagerFactoryUtil.shutdown.
- Code Content:**

```
import com.github.javafaker.Faker;
import dao.*;
import entities.*;

import javax.persistence.EntityManager;
import java.util.Date;
import java.util.HashSet;
import java.util.Set;

import static entities.enums.MatchEventType.INJURY;
import static entities.enums.MatchEventType.YELLOW_CARD;
import static entities.enums.MatchEventType.RED_CARD;
import static util.EntityManagerFactoryUtil.createEntityManagerFactory;
import static util.EntityManagerFactoryUtil.shutdown;

public class App {
    static UserDao userDao;
    static PlayerDao playerDao;
    static CoachDao coachDao;
    static CityDao cityDao;
    static CompetitionDao competitionDao;
    static ContractDao contractDao;
    static MatchEventDao matchEventDao;
    static StadiumDao stadiumDao;
    static TeamDao teamDao;
    static Faker faker = new Faker();

    public static void main(String[] args) {
        EntityManagerFactory entityManagerFactory = createEntityManagerFactory();
        EntityManager entityManager = entityManagerFactory.createEntityManager();
        entityManager.getTransaction().begin();
        entityManager.persist(user);
        entityManager.getTransaction().commit();
        entityManager.close();
        entityManagerFactory.close();
    }
}
```
- Toolbars and Status Bar:** The status bar at the bottom shows 8 chars, 36:44, CRLF, UTF-8, 4 spaces, and the date/time 6:10 PM 1/10/2021.

The screenshot shows a Java application named "footballGame" in an IDE. The main class is "App.java". The code performs various database operations using Entity Managers and DAOs.

```
public static void main(String[] args) {
    EntityManager entityManager = createEntityManagerFactory().createEntityManager();
    daoInitialization(entityManager);
    entityManager.getTransaction().begin();

    // گرانترین مردی به همراه دستمزد آن
    coachDao.expensiveCoach().forEach(System.out::println);

    // فهرست گران ترین بازیکن در هر فصل
    playerDao.expensivePlayerInSeason().forEach(System.out::println);

    // فهرست شهر ها و تعداد تیم های آن ها
    cityDao.numberOfTeam();

    // برای سال 2020 مجموع امتیازهای هر تیم در خانه حربه
    teamDao.homeScoreInYear(2020)
        .forEach(objects -> System.out.println("team:" + objects[0] + " score: " + objects[1]));

    // برای سال 2020 مجموع امتیازهای هر تیم در خارج از خانه حربه
    teamDao.awayScoreInYear(2020)
        .forEach(objects -> System.out.println("team:" + objects[0] + " score: " + objects[1]));

    // برای سال 2020 مجموع امتیازهای هر تیم
    teamDao.scoresInYear(2020).entrySet()
        .forEach(entrySet -> System.out.println("team: "+entrySet.getKey() + " score: "+entrySet.getValue()));

    // 2020 فهرمان سال
    teamDao.heroOfYear(2020);
}
```

This screenshot shows the same Java application "footballGame" in the IDE, with the code for the "App.java" class. The code is identical to the one in the first screenshot, performing database operations like calculating home and away scores for teams in 2020.

```
teamDao.awayScoreInYear(2020)
    .forEach(objects -> System.out.println("team:" + objects[0] + " score: " + objects[1]));

    // برای سال 2020 مجموع امتیازهای هر تیم
    teamDao.scoresInYear(2020).entrySet()
        .forEach(entrySet -> System.out.println("team: "+entrySet.getKey() + " score: "+entrySet.getValue()));

    // 2020 فهرمان سال
    teamDao.heroOfYear(2020);

    entityManager.getTransaction().commit();
    entityManager.close();
    shutdown();
}

public static void daoInitialization(EntityManager entityManager) {
    userDao = new UserDao(entityManager);
    playerDao = new PlayerDao(entityManager);
    coachDao = new CoachDao(entityManager);
    competitionDao = new CompetitionDao(entityManager);
    cityDao = new CityDao(entityManager);
    contractDao = new ContractDao(entityManager);
    matchEventDao = new MatchEventDao(entityManager);
    stadiumDao = new StadiumDao(entityManager);
    teamDao = new TeamDao(entityManager);
}
```

coachDao:

The screenshot shows the CoachDao.java file in an IDE. The code implements the UserDao interface and extends EntityManager. It uses CriteriaQuery to select users who are coaches (not players). A note in Persian at the bottom left indicates that the code retrieves the maximum salary for coaches.

```
import entities.Coach;
import entities.Contract;
import entities.Player;
import entities.User;
import javax.persistence.EntityManager;
import javax.persistence.TypedQuery;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Join;
import javax.persistence.criteria.Root;
import java.util.List;

public class CoachDao extends UserDao {

    public CoachDao(EntityManager entityManager) { super(entityManager); }

    // جون از ارث برای برای بازیکن و مرتبی استفاده شده‌است این سند برای پوزر هایی که فقط مرتبی هستند استفاده می‌کنند
    public List<User> coaches() {
        //main criteria query that return list of users
        CriteriaQuery<User> criteria = getCriteriaBuilder().createQuery(User.class);
        // root for select in user
        Root<User> fromUser = criteria.from(User.class);

        //users that types is coach(not player)
        criteria.select(fromUser).where(getCriteriaBuilder().notEqual(fromUser.type(), Player.class));

        TypedQuery<User> typedQuery = entityManager.createQuery(criteria);
        return typedQuery.getResultList();
    }

    // این سند بیشترین قیمت قراردادی که تا به حال برای یک مرتبی پسته شده است را نشان می‌دهد
    public List<Double> maxSalary() {
    }
}
```

Event Log: 39 chars, 1 line break, 57:26, CRLF, UTF-8, 4 spaces

The screenshot shows the CoachDao.java file in an IDE with code completion enabled. The cursor is on the 'join' keyword, which is underlined in blue. A tooltip or completion dropdown is visible, showing options related to the current context.

```
// این سند بیشترین قیمت قراردادی که تا به حال برای یک مرتبی پسته شده است را نشان می‌دهد
public List<Double> maxSalary() {
    //main criteria query that return list of coach salary
    CriteriaQuery<Double> criteria = getCriteriaBuilder().createQuery(Double.class);
    // root for select in contract
    Root<Contract> fromContract = criteria.from(Contract.class);

    //extracts max of salary for coach
    criteria.select(getCriteriaBuilder().max(fromContract.get("salary")))
        .where(fromContract.get("user").in(coaches()));

    TypedQuery<Double> typedQuery = entityManager.createQuery(criteria);
    return typedQuery.getResultList();
}

// این سند گرون ترین مرتبی را نشان می‌دهد
public List<Coach> expensiveCoach() {
    //main criteria query that return list of coaches
    CriteriaQuery<Coach> criteria = getCriteriaBuilder().createQuery(Coach.class);
    // root for select in coach
    Root<Coach> fromCoach = criteria.from(Coach.class);
    //join with coach and contract
    Join<Coach, Contract> contractJoin = fromCoach.join("contracts");

    //select coach that salary is equal to (salary of expensive coach)
    criteria.select(fromCoach).where(contractJoin.get("salary").in(maxSalary()));

    TypedQuery<Coach> typedQuery = entityManager.createQuery(criteria);
    return typedQuery.getResultList();
}
```

Event Log: 39 chars, 1 line break, 57:26, CRLF, UTF-8, 4 spaces

playerDao:

The screenshot shows the code for the `PlayerDao` class in the `footballGame` project. The code retrieves a list of users who are players (not coaches) from the database.

```
import entities.Coach;
import entities.Contract;
import entities.Player;
import entities.User;

import javax.persistence.EntityManager;
import javax.persistence.TypedQuery;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Join;
import javax.persistence.criteria.Root;
import java.util.List;

public class PlayerDao extends UserDao{

    public PlayerDao(EntityManager entityManager) { super(entityManager); }

    // ليست بوزر هایی که بازیکن هستند
    public List<User> players() {
        //main criteria query that return list of users
        CriteriaQuery<User> criteria = getCriteriaBuilder().createQuery(User.class);
        // root for select in user
        Root<User> fromUser = criteria.from(User.class);

        //users that types is player(not coach)
        criteria.select(fromUser).where(getCriteriaBuilder().notEqual(fromUser.type(), Coach.class));

        TypedQuery<User> typedQuery = entityManager.createQuery(criteria);
        return typedQuery.getResultList();
    }

    // ليست پیشترین مسترد در هر فصل
    public List<Double> maxSalaryInSeason();
}
```

The screenshot shows the implementation of two methods: `maxSalaryInSeason()` and `expensivePlayerInSeason()`. The `maxSalaryInSeason()` method retrieves the maximum salary for players grouped by season. The `expensivePlayerInSeason()` method retrieves players whose salary is the maximum for their respective seasons.

```
// ليست پیشترین مسترد در هر فصل
public List<Double> maxSalaryInSeason() {
    //main criteria query that return list salary of players
    CriteriaQuery<Double> criteria = getCriteriaBuilder().createQuery(Double.class);
    // root for select in contract
    Root<Contract> fromContract = criteria.from(Contract.class);

    //max salary where user is player and group by season
    criteria.select(getCriteriaBuilder().max(fromContract.get("salary")))
        .where(fromContract.get("user").in(players()))
        .groupBy(fromContract.get("season"));

    TypedQuery<Double> typedQuery = entityManager.createQuery(criteria);
    return typedQuery.getResultList();
}

// این متد گران ترین بازیکن هر فصل را نشان میدهد
public List<Player> expensivePlayerInSeason() {
    //main criteria query that return list of players
    CriteriaQuery<Player> criteria = getCriteriaBuilder().createQuery(Player.class);
    // root for select in players
    Root<Player> fromPlayer = criteria.from(Player.class);
    //join with player and contract
    Join<Player,Contract> contractJoin = fromPlayer.join("contracts");

    //players that salary in max salary
    criteria.select(fromPlayer).where(contractJoin.get("salary").in(maxSalaryInSeason()));

    TypedQuery<Player> typedQuery = entityManager.createQuery(criteria);
    return typedQuery.getResultList();
}
```

cityDao:

The screenshot shows the IntelliJ IDEA interface with the CityDao.java file open in the editor. The code implements the CityDao interface, extending AbstractDao<Integer, City>. It overrides the getEntityClass() method to return City.class. The numberOfTeam() method uses CriteriaQuery to count the number of teams per city. The typedQuery.getResultSet() method prints the results to the console.

```
import entities.City;
import entities.Team;

import javax.persistence.EntityManager;
import javax.persistence.TypedQuery;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Join;
import javax.persistence.criteria.Root;
import java.util.List;

public class CityDao extends AbstractDao<Integer, City> {

    public CityDao(EntityManager entityManager) { super(entityManager); }

    @Override
    public Class<City> getEntityClass() { return City.class; }

    //تعداد تیم های هر شهر را به همراه نام شهر چاپ میکند
    public void numberofTeam() {
        CriteriaQuery<Object[]> criteria = getCriteriaBuilder().createQuery(Object[].class);
        Root<Team> fromTeam = criteria.from(Team.class);
        //multiselect => city name, count(team of city) => group by city
        criteria.multiselect(fromTeam.get("city").get("cityName"), getCriteriaBuilder().count(fromTeam))
            .groupBy(fromTeam.get("city").get("cityName"));

        TypedQuery<Object[]> typedQuery = entityManager.createQuery(criteria);
        typedQuery.getResultList()
            .forEach(objects -> System.out.println("city: "+objects[0]+", NumOfTeam: "+ objects[1]));
    }
}
```

teamDao:

The screenshot shows the IntelliJ IDEA interface with the TeamDao.java file open in the editor. The code implements the TeamDao interface, extending AbstractDao<Integer, Team>. It overrides the getEntityClass() method to return Team.class. The homeScoreInYear() method uses CriteriaQuery to calculate the total score of home teams for a specific year. The typedQuery.getResultSet() method prints the results to the console.

```
import java.util.List;
import java.util.Map;

public class TeamDao extends AbstractDao<Integer, Team> {

    public TeamDao(EntityManager entityManager) { super(entityManager); }

    @Override
    public Class<Team> getEntityClass() { return Team.class; }

    //مجموع امتیازی که تیم ها را در سال دلخواه در خانه کسب کرده اند به صورت مرتب شده بر اساس نام تیم میباشد
    public List<Object[]> homeScoreInYear(Integer year) {
        //main criteria query that return list of arrays
        CriteriaQuery<Object[]> criteria = getCriteriaBuilder().createQuery(Object[].class);
        // root for select in Competition
        Root<Competition> fromCompetition = criteria.from(Competition.class);

        //multiselect => home team name, sum of score in home in special year
        criteria.multiselect(fromCompetition.get("homeTeam").get("teamName")
            , getCriteriaBuilder().sum(fromCompetition.get("homeScore")))
            .where(getCriteriaBuilder().equal(fromCompetition.get("season"), year))
            //if dont group by, query sum of all home team scores(ex:2 ,3 ,1 => 6)
            .groupBy(fromCompetition.get("homeTeam").get("teamName"))
            //order bu team names to can use it for future
            .orderBy(getCriteriaBuilder().asc(fromCompetition.get("homeTeam").get("teamName")));

        TypedQuery<Object[]> typedQuery = entityManager.createQuery(criteria);
        return typedQuery.getResultList();
    }
}
```

مجموع امتیازی که تیم ها را در سال دلخواه در زمین حریف کسب کرده اند به همراه اسم تیم به صورت مرتب شده بر اساس ثامن تیم مددهد

```
public List<Object[]> awayScoreInYear(Integer year) {
    //main criteria query that return list of arrays
    CriteriaQuery<Object[]> criteria = getCriteriaBuilder().createQuery(Object[].class);

    // root for select in Competition
    Root<Competition> fromCompetition = criteria.from(Competition.class);

    //multiselect => away team name, sum of score in home in special year
    criteria.multiselect(fromCompetition.get("awayTeam").get("teamName")
        , getCriteriaBuilder().sum(fromCompetition.get("awayScore")))
        .where(getCriteriaBuilder().equal(fromCompetition.get("season"), year))
        //if dont group by,query sum of all away team scores(ex:6 ,2 ,1 => 9)
        .groupBy(fromCompetition.get("awayTeam").get("teamName"))
        //order bu team names to can use it for future
        .orderBy(getCriteriaBuilder().asc(fromCompetition.get("awayTeam").get("teamName")));

    TypedQuery<Object[]> typedQuery = entityManager.createQuery(criteria);
    return typedQuery.getResultList();
}

ابن متد همی از تیم هارو به همراه اسم تیم و امتیاز تیم ها در سال خاصی نشان میدهد
public Map<String, Long> scoresInYear(Integer year) {
    //list of teams name with they're scores in home and away
    List<Object[]> homeScore = homeScoreInYear(year);
    List<Object[]> awayScore = awayScoreInYear(year);
    // map with team name and (home+away) score
    Map<String, Long> teamScores = new HashMap<>();
```

```
TypedQuery<Object[]> typedQuery = entityManager.createQuery(criteria);
return typedQuery.getResultList();

ابن متد همی از تیم هارو به همراه اسم تیم و امتیاز تیم ها در سال خاصی نشان میدهد
public Map<String, Long> scoresInYear(Integer year) {
    //list of teams name with they're scores in home and away
    List<Object[]> homeScore = homeScoreInYear(year);
    List<Object[]> awayScore = awayScoreInYear(year);
    // map with team name and (home+away) score
    Map<String, Long> teamScores = new HashMap<>();

    for (int i = 0; i < awayScore.size(); i++) {
        teamScores.put((String) homeScore.get(i)[0], ((Long) awayScore.get(i)[1] + (Long) homeScore.get(i)[1]));
    }
    return teamScores;
}

ابن متد اسم تیم قیفرمان به همراه امتیاز ان را در سال خاصی نشان میدهد
public void heroOfYear(Integer year) {
    Map<String, Long> teamScores = scoresInYear(year);
    //map that store one element => hero team name, score of it
    Map.Entry<String, Long> max = Collections.max(teamScores.entrySet(), (e1, e2) -> e1.getValue().compareTo(e2));
    System.out.println("Hero Name: " + max.getKey() + " Score is: " + max.getValue());
}
```

تمرین شماره ۲:

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help Emp_Add_Pho - App.java

Project Emp_Add_Pho C:\Users\Soheil\IdeaProjects\Emp_Add_Pho src main java App main

```

import static util.Factory.*;
import javax.persistence.EntityManager;

public class App {
    static Faker faker = new Faker();
    public static void main(String[] args) {
        EntityManager entityManager = createEntityManagerFactory().createEntityManager();
        MyQuery myQuery = new MyQuery(entityManager);
        entityManager.getTransaction().begin();

        // شهر با بیشترین مردم
        myQuery.mostSalaryByCity()
            .forEach(objects -> System.out.println("city: " + objects[1] + " salary: " + objects[0]));

        // بیشترین درآمد کارمندان هر شهر
        myQuery.empMostSalaryByCity()
            .forEach(objects -> System.out.println(objects[0] + " city: " + objects[1]));

        // جست و جوی کارمند بر اساس کد پستی
        System.out.println(myQuery.searchByPostalCode("212343"));

        // جست و جوی کارمند بر اساس شماره تلفن
        System.out.println(myQuery.searchByTelNumber("124200"));

        entityManager.getTransaction().commit();
        entityManager.close();
        shutdown();
    }
}

```

Lombok Requires Annotation Processing: Do you want to enable annotation processors? Enable

Event Log

Type here to search

30-3 CRLF UTF-8 4 spaces

6:21 PM 1/10/2021

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help Emp_Add_Pho - MyQuery.java

Project Emp_Add_Pho C:\Users\Soheil\IdeaProjects\Emp_Add_Pho src main java util MyQuery searchByTelNumber

```

import entities.Phone;
import javax.persistence.EntityManager;
import javax.persistence.TypedQuery;
import javax.persistence.criteria.*;
import java.util.List;
import java.util.stream.Collectors;

public class MyQuery {
    private EntityManager entityManager;
    private CriteriaBuilder builder;

    public MyQuery(EntityManager entityManager) {
        this.entityManager = entityManager;
        this.builder = entityManager.getCriteriaBuilder();
    }

    public List<Object> mostSalaryByCity() {
        CriteriaQuery<Object[]> criteria = builder.createQuery(Object[].class);
        Root<Employee> fromEmployee = criteria.from(Employee.class);
        Join<Employee, Address> employeeAddressJoin = fromEmployee.join("addresses");

        // multiselect => max salary, city of it => group by city
        criteria.multiselect(builder.max(fromEmployee.get("salary"))
            ,employeeAddressJoin.get("city"))
            .groupBy(employeeAddressJoin.get("city"));

        TypedQuery<Object[]> typedQuery = entityManager.createQuery(criteria);
        return typedQuery.getResultList();
    }
}

```

Lombok Requires Annotation Processing: Do you want to enable annotation processors? Enable

Event Log

Type here to search

70:54 CRLF UTF-8 4 spaces

6:22 PM 1/10/2021

این متد کارمندی که بیشترین درامد را در هر شهر دارد نمایندگاند //

```
public List<Object[]> empMostSalaryByCity() {
    //extract max salary for employees in city
    List<Double> maxSalary = mostSalaryByCity();
    .stream()
    .map(objects -> (Double)objects[0])
    .collect(Collectors.toList());
}

CriteriaQuery<Object[]> criteria = builder.createQuery(Object[].class);
Root<Employee> fromEmployee = criteria.from(Employee.class);
Join<Employee, Address> employeeAddressJoin = fromEmployee.join("addresses");

//multiselect employee ,city where salary is maximum
criteria.multiselect(fromEmployee, employeeAddressJoin.get("city"))
    .where(fromEmployee.get("salary").in(maxSalary));

TypedQuery<Object[]> typedQuery = entityManager.createQuery(criteria);
return typedQuery.getResultList();
}
```

این متد کارمند را بر اساس کدیستی آن پیدا میکند//

```
public Employee searchByPostalCode(String postalCode) {
    CriteriaQuery<Employee> criteria = builder.createQuery(Employee.class);
    Root<Employee> fromEmployee = criteria.from(Employee.class);
    Join<Employee, Address> employeeAddressJoin = fromEmployee.join("addresses");

    criteria.select(fromEmployee).where(builder.equal(employeeAddressJoin.get("postalCode"), postalCode));

    TypedQuery<Employee> typedQuery = entityManager.createQuery(criteria);
    return typedQuery.getSingleResult();
}
```

}

این متد کارمند را بر اساس کدیستی آن پیدا میکند//

```
public Employee searchByPostalCode(String postalCode) {
    CriteriaQuery<Employee> criteria = builder.createQuery(Employee.class);
    Root<Employee> fromEmployee = criteria.from(Employee.class);
    Join<Employee, Address> employeeAddressJoin = fromEmployee.join("addresses");

    criteria.select(fromEmployee).where(builder.equal(employeeAddressJoin.get("postalCode"), postalCode));

    TypedQuery<Employee> typedQuery = entityManager.createQuery(criteria);
    return typedQuery.getSingleResult();
}
```

این متد کارمند را بر اساس شماره تلفن پیدا میکند//

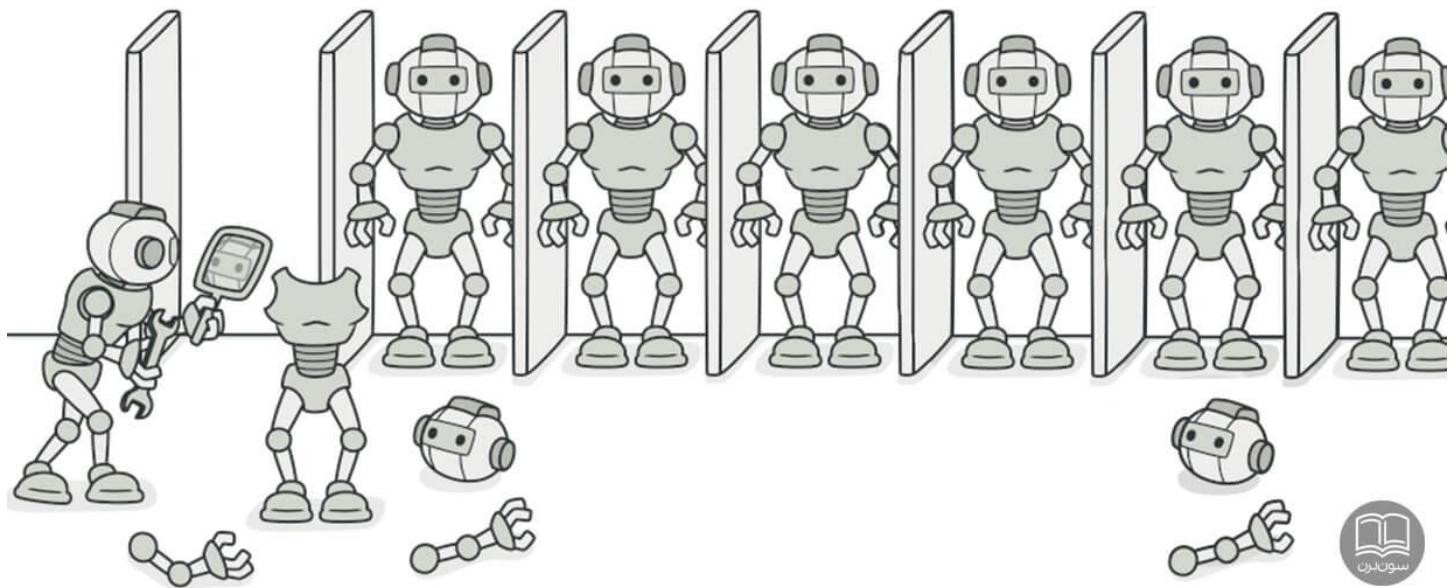
```
public Employee searchByTelNumber(String telNumber) {
    CriteriaQuery<Employee> criteria = builder.createQuery(Employee.class);
    Root<Employee> fromEmployee = criteria.from(Employee.class);
    Join<Employee, Address> employeeAddressJoin = fromEmployee.join("addresses");
    Join<Address, Phone> addressPhoneJoin = employeeAddressJoin.join("phones");

    criteria.select(fromEmployee).where(builder.equal(addressPhoneJoin.get("telNumber"), telNumber));

    TypedQuery<Employee> typedQuery = entityManager.createQuery(criteria);
    return typedQuery.getSingleResult();
}
```

الگوی پروتوتایپ (Prototype)

دیزاین پترن ها در سال ۱۹۹۴ توسط گروهی به نام Gang of Four در سه دسته عمومی طبقه بندی شدند. به آن دسته از الگوهای طراحی که با هدف مدیریت ایجاد اشیا توسعه یافته اند، الگوهای طراحی سازنده یا Creational می‌گویند. الگوی طراحی سازنده Prototype یکی از الگوهای طراحی سازنده است که به منظور جلوگیری از ساخت یک شی جدید استفاده می‌شود.

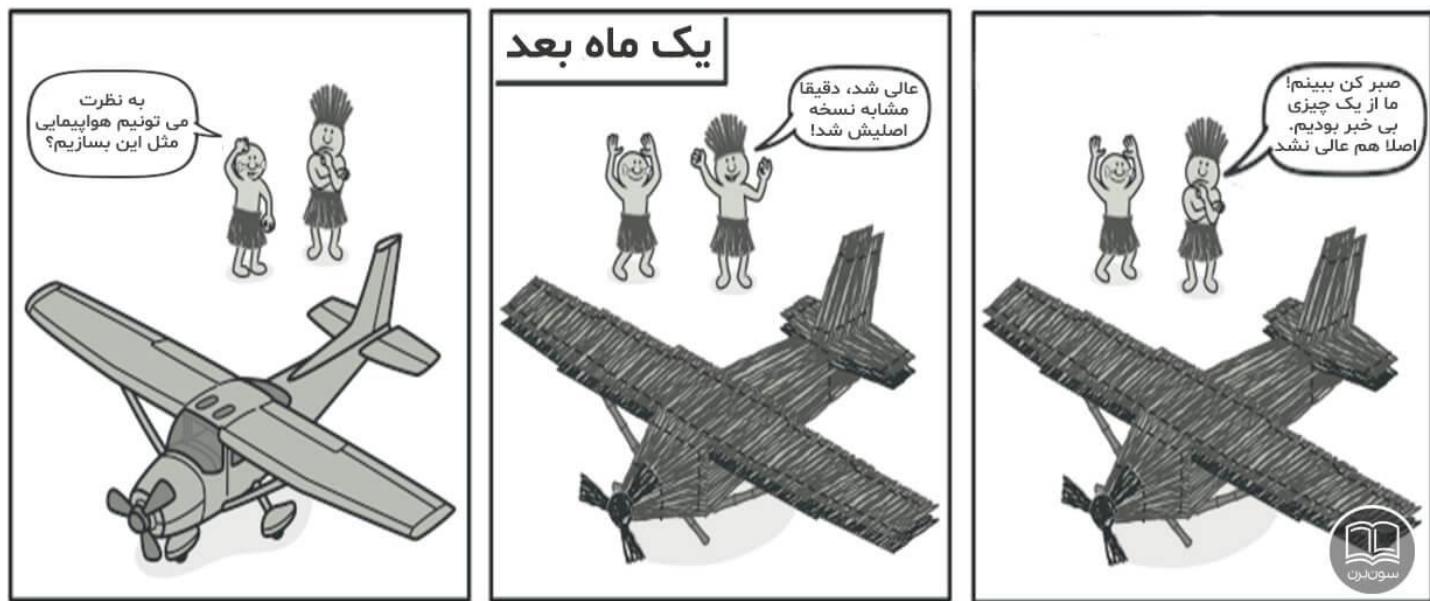


این الگوی طراحی راهکاری برای ایجاد یک شی جدید به واسطه کپی کردن آن از اشیا موجود دیگر ارائه می‌دهد. با استفاده از این تکنیک بدون اینکه وابستگی میان شی جدید و کلاسی که از آن ساخته شده است ایجاد شود، می‌توان اشیا جدیدی در نرم افزار تولید کرد. همچنین در مواردی که موقع نیاز به ایجاد تغییرات در اشیا جدید وجود دارد تا بتوانیم اشیایی از یک نوع و با پارامترهایی متفاوت ایجاد کنیم.

مثلاً تصور کنید بخواهید از یک شی ماشین چندین کپی ایجاد کنید با این تفاوت که هر کدام رنگی متفاوت داشته باشد. با استفاده از الگوی طراحی پروتوتایپ دیگر نیاز نیست که هر بار یک شی جدید از کلاس ایجاد کنید و سپس پارامترهای آن را مشخص کنید. زیرا شما می‌توانید از کلاس ماشین یک شی ایجاد کنید و سپس از آن به تعدادی که نیاز دارید کپی (Clone) تولید کنید.

چرا باید از الگوی طراحی Prototype استفاده کنیم

تصور کنید که تصمیم گرفته اید یک هواپیما بر اساس نمونه خارجی آن بسازید. برای بومی ساز این هواپیما تنها راهی که پیش روی شما خواهد بود این است که دقیقاً از پارامترها و ویژگی‌های ساختاری نمونه خارجی آن آگاه باشید. اما از آنجایی که تکنولوژی‌هایی مربوط به ساخت هواپیماها انحصاری هستند و به راحتی در اختیار دیگر شرکت‌ها قرار نمی‌گیرند، این امر برای شما امکان پذیر نخواهد بود. اما تصویر کنید که یک ماشین کپی سه بعدی غول پیکر برای شبیه سازی اشیا داشته باشد. با این کار می‌توانید هواپیما خارجی به آن دستگاه وارد کرده و هواپیما شبیه سازی خود را دریافت کنید. این ماشین شبیه ساز کاری شبیه الگوی طراحی پرووتایپ انجام می‌دهد.



بنابراین یکی از مشکلات کپی کردن اشیا در برنامه نویسی به صورت مستقیم، عدم دسترسی به برخی از پارامترهای خصوصی است. پس کپی کردن یک شی خارج از کلاس همیشه امکان پذیر نخواهد بود. از طرفی دیگر به دلیل اینکه شما اشیای کپی شده را بر پایه کلاس‌های شی مبدأ ایجاد می‌کنید، اشیای جدید به این کلاس‌ها وابسته می‌شوند و این کار باعث ایجاد وابستگی‌های زیاد در بین کلاس‌ها خواهد شد.

در الگوی طراحی نمونه اولیه یک Interface مشترک برای تمام اشیایی که باید کپی شوند در نظر گرفته می‌شود. این Interface ابه شما اجازه می‌دهد که یک شی را بدون نیاز به دوباره نویسی کدها در کلاس شی، کپی کنید. معمولاً، چنین Interface شامل تنها یک تابع به اسم Clone (کپی) هستند. پس با پیروی کلاس‌های مربوط به اشیا از این Interface دیگر نیازی به نوشتن توابع اضافه و یا استفاده از ایجاد مجدد یک شی نخواهیم داشت. برای این کار فقط کافی است تابع Clone را در کلاس‌ها پیاده سازی کنیم.

پیاده سازی تابع Clone در همه کلاس‌ها بسیار شبیه به هم است. این تابع وظیفه دارد تا یک شی از کلاس فعلی ایجاد کند و سپس تمام مقادیر پارامترهای آن را به شی جدید منتقل می‌کند. همچنین مزیت دیگری که این الگوی طراحی در اختیار شما می‌گذارد این است که حتی می‌توانید پارامترهای Private را کپی کنید. زیرا اکثر زبان‌های برنامه نویسی اجازه می‌دهند که اشیاء به فیلدهای خصوصی از اشیاء دیگر که متعلق به یک کلاس هستند دسترسی پیدا کنند.

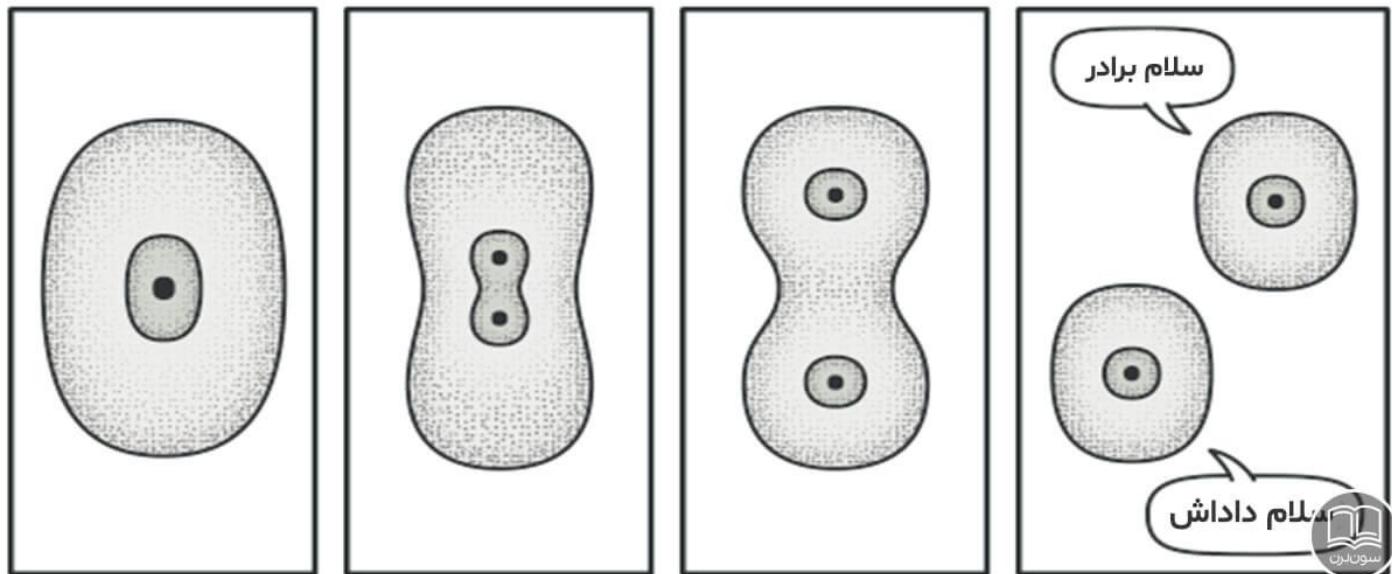
به اشیایی که کلاس‌های آن‌ها از قابلیت کپی شدن پشتیبانی کنند یک نمونه اولیه یا Prototype نامیده می‌شوند. همچنین هنگامی که اشیا دارای دهها پارامتر و صدها تنظیمات باشند، می‌توانید از اشیا کپی شده به عنوان جایگزینی برای کلاس‌های فرزند استفاده شوند. با این کار در نرم افزار دیگر نیازی به ساخت کلاس‌های فرزند از کلاس اصلی نخواهید داشت. زیرا می‌توانید اشیا ساخته شده بر اساس کلاس اصلی را متناسب با نیازهایی که دارد تغییر دهید.



بنابراین اگر بخواهیم روش کار این الگو را جمع بندی کنیم باید گفت که شما در ابتدا باید مجموعه‌ای از اشیا را ایجاد کنید که به شیوه‌های مختلف پیکربندی شده‌اند. سپس در هر زمانی که به یک شی با یکی از پیکربندی‌های تنظیم شده نیاز داشته باشید، کافی است یک نمونه اولیه را بر اساس آن ایجاد کنید و دیگر نیازی به ساخت یک شی جدید از ابتدا نخواهد داشت.

مثالی از الگوی طراحی پروتوتایپ در دنیای واقعی

در دنیای واقعی، نمونه‌های اولیه برای انجام آزمایش‌های مختلف قبل از شروع تولید انبوه مخصوصات استفاده می‌شوند. اما در این الگوی طراحی، نمونه‌های اولیه به منظور انجام آزمایشات مختلف یا کنترل کیفیت استفاده نمی‌شوند. نمونه‌های ساخته شده در این تکنیک به صورت مستقیم در بخش‌های مورد نیاز در نرم افزار استفاده می‌شوند. بنابراین، تقریباً بهترین مثال از این الگوی طراحی در دنیای واقعی فرآیند تقسیم سلولی است.



تصور کنید که یک کلاس به اسم سلول (Cell) در نرم افزار تعریف شود. برای اینکه این سلول به یک موجود تکامل یافته تبدیل شود، نیاز به تکثیر و نمونه سازی دارد. بنابراین به هزاران یا میلیون‌ها شی از این کلاس نیاز خواهیم داشت تا اندام‌های مختلف یک موجود زنده را تشکیل دهند و هر کدام به وظیفه خاص خود بپردازد. حال تصور کنید، اگر می‌خواستید همه این اشیا را به صورت مستقیم و یک به یک بسازید و پارامترهای آن‌ها را مقدار دهی کنید، چه میزان از زمان شما مصرف می‌شد.

بر این اساس بهترین راه برای انجام این کار، به کارگیری الگوی طراحی پروتوتایپ خواهد بود. زیرا در این الگو سلول‌ها بر اساس سلول اولیه کپی خواهند شد و دیگر نیازی به ساخته شدن سلول‌ها و مقدار دهی پارامترهای آن‌ها بر اساس سلول اولیه به صورت یک به یک نخواهد بود. با توجه به این نکته که در فرآیند تقسیم سلولی، سلول‌های جدید از سلول اولیه ایجاد می‌شوند می‌توان اینگونه گفت که سلول اصلی به عنوان یک نمونه اولیه یا Prototype عمل می‌کند.



به کارگیری الگوی طراحی پروتوتایپ در طراحی نرم افزارها، باعث کاهش میزان کدنویسی خواهد شد. مزیت اصلی این دیزاین پترن افزایش سرعت کپی کردن یک شی از شی‌های است. زیرا این عمل خیلی سریع‌تر از ساخت اشیا می‌باشد. در عملیات کپی کردن اشیا تابع سازنده آن‌ها (Constructor) دیگر اجرا نخواهد شد. الگوی طراحی سازنده در موارد زیر کاربرد خواهد داشت:

- ۱- برای کاهش تعداد کلاس‌های فرزند (Subclass) استفاده می‌شود. همچنین با این روش می‌توان اشیا ساخته شده را متناسب با نیاز‌های مختلف پیکربندی کرد.
- ۲- به منظور جلوگیری از ایجاد وابستگی میان شی کپی شده به کلاس مورد نظر استفاده می‌شود.
- ۳- زمانی که بخواهیم از یک کلاس با پارامترهای Private زیاد شی ایجاد کنیم.