Sinem Gündüzalp

20200805004

1) Unit test yazılımdaki en küçük birimin (bir method veya method içindeki bi parça kod vs) diğer birimlerden bağımsız olarak doğru bir şekilde çalışıp çalışmadığını kontrol eder. bu test var olan hatayı eken farketmemizi sağlar, kodun doğru çalıştığına dair güvenilirliği artırır,kodda karmaşıklığı azaltıp kaliteyi artırır, istenmeyen değişikliklerden korur geliştirdiğimiz özellikleri unit test ile korumaya alabiliriz, unit testine bakarak o kodun nasıl çalıştığı hakkında bilgi edinebiliriz, debug ile tek tek breakpoint yapmak yerine hızlıca debug yapmış gibi oluruz. Unit test için birden fazla Framework vardır bunlardan bazıları; JUnit, TestNG, Mockito, PowerMock ve Spock ama en kullanılan framework JUnit'tir. Aynı zamanda Unit test yaparken Uymamız gereken kurallar vardır. Her test birbirinden bağımsız olmalı, testler tekrarlandığında her zaman aynı sonucu vermeli ve tutarlı olmalı, hızlı yapılmalı çok vaktimizi almamalı, anlaşılır olmalı, birden fazla durumu aynı anda test etmek yerine, her testin belirli bir özelliği veya işlevi test etmeli, giriş değerleri için sınır koşullarını kontrol etmeli ve limit durumları da dahil etmeli, testlerin isimlendirmesi açık ve tanımlayıcı olmalı, yapılan değişiklikler sonrasında hala doğru sonuçlar üretmeli, kodun karmaşıklığını azaltmak için uygun bir yapı ve gruplama kullanılmalı bu kurallara uyarak yazılan unit testler kodun güvenirliliğini artırır.

2) Design Patterns, yerleşik bir düşünme tekniğidir. Dilden bağımsızdır. Halihazırda yapılmış olanı yapmaktan kaçınmamızı sağlar. Yaygın olarak karşılaşılan zorluklar için yeniden kullanılabilir nesne yönelimli yazılımlardır. Ve de defalarca test edildiğinden güvenilirdir.

Tasarım kalıpları 3 çeşittir.

- Creational Patterns (Oluşturucu Desenler): Yeni nesnelerin nasıl oluşturulacağını ele alır. Örnekler arasında Prototype, Singleton, Factory, Abstract Factory ve Builder vardır.

- Structural Patterns (Yapısal Desenler): Nesneler arasındaki ilişkileri nasıl organize edeceğimizi ele alır. Örnekler arasında Adapter, Composite, Proxy, Fly Weight, Facade, Decorator ve Bridge vardır.

- Behavioral Patterns (Davranışsal Desenler): Nesnelerin birbirleriyle nasıl iletişim kuracaklarını ele alır. Örnekler arasında Template Method, Observer, Strategy, Mediator, Visitor, Iterator ve Command Vardır.

3)
```
import java.util.Random;
import javax.swing.JOptionPane;
public class PasswordGenerator {
    public static void main(String[] args) {
        String password = generatePassword();
        JOptionPane.showMessageDialog(null, "Oluşturduğunuz Şifre: " + password);
    }
    public static String generatePassword() {
        String specialCharacters = "!@#$%^*()~`{}]\\&_";
        Random random = new Random();
        while (true) {
            StringBuilder password = new StringBuilder();
            int length = 8 + random.nextInt(87);
            password.append((char) (97 + random.nextInt(26)));
            password.append(random.nextInt(10));
            for (int i = 2; i < length; i++) {
                char randomChar = (char) (32 + random.nextInt(94));
                password.append(randomChar);
            }
            if (isValidPassword(password.toString(), specialCharacters)) {
                return password.toString();
            }
        }
    }
    public static boolean isValidPassword(String password, String specialCharacters) {
        return password.length() >= 8 &&
                password.length() <= 95 &&
                password.matches(".*[a-zA-Z]+.*") &&
                password.matches(".*\\d+.*") &&
                !password.startsWith(" ") &&
                !password.endsWith(" ") &&
                password.matches(".*[" + specialCharacters + "]+.*");
    }
}
```

4)---

2.4A)

```
public class PerfectNumber {

    public static boolean isPerfect(int number) {
        if (number <= 0) {
            return false;
        }

        int sum = 0;
        for (int i = 1; i < number; i++) {
            if (number % i == 0) {
                sum += i;
            }
        }

        return sum == number;
    }

    public static void main(String[] args) {
        int num = 28; // Test etmek istenilen sayı buraya girilir
        boolean result = isPerfect(num);

        if (result) {
            System.out.println(num + " mükemmel bir sayı.");
        } else {
            System.out.println(num + " mükemmel bir sayı değil.");
        }
    }
}
```

2.4B)

```
public class PerfectNumber {

    public static boolean isPerfect(int number) {
        if (number <= 0) {
            return false;
        }

        int sum = 0;
        for (int i = 1; i < number; i++) {
            if (number % i == 0) {
                sum += i;
            }
        }

        return sum == number;
    }

    public static void main(String[] args) {
        for (int i = 1000; i <= 9999; i++) {
            if (isPerfect(i)) {
                System.out.println("4 basamaklı mükemmel sayı= " + i);
            }
        }
    }
}
```

2.2)

```
public class SumFactors {

    public static int sumFactors(int number) {
        int sum = 0;

        for (int i = 1; i <= number / 2; i++) {
            if (number % i == 0) {
                sum += i;
            }
        }

        return sum;
    }

    public static void main(String[] args) {
        int num = 28; // Test etmek istediğiniz sayıyı buraya gir
        int result = sumFactors(num);

        System.out.println(num + " sayısının çarpanlarının toplamı: " + result);
    }
}
```

2.3)

```
public class ArkadasSayilar {
    public static boolean areFriends(int x, int y) {
        int topX = 0
        int topY = 0;
        // x'in bölenlerinin toplamı
        for (int i = 1; i <= x / 2; i++) {
            if (x % i == 0) {
                topX += i;
            }
        }
        // y'nin bölenlerinin toplamı
        for (int i = 1; i <= y / 2; i++) {
            if (y % i == 0) {
                topY += i;
            }
        }

        return topX == y && topY == x;
    }

    public static void main(String[] args) {
        for (int i = 1000; i < 10000; i++) {
            for (int j = i + 1; j < 10000; j++) {
                if (areFriends(i, j)) {
                    System.out.println("Arkadaş sayı çifti= " + i + " ve " + j);
                    return;
                }
            }
        }
    }
}
```