İLERİ JAVA UYGULAMALARI
FİNAL ÖDEVİ

AD: Şevval

SOYAD: Karagöz

OKUL NO: H5220039

DERS ADI: İleri Java Uygulamaları
KONUSU: Değişkenler ve Veri Tipleri

Değişkenler

Değişken Tanımı ve Özellikleri

Değişken Tanımı: Değişkenler, programlarımızda verileri saklamak için kullandığımız isimlendirilmiş alanlardır. Bir kutuya benzetebiliriz; kutunun içine veri koyarız ve bu kutuya bir isim veririz.

Özellikleri:

1. **Tip:** Değişkenler, sakladıkları verinin türüne göre farklı tiplerde olabilir. Örneğin, sayılar için int, metinler için String, ondalıklı sayılar için double, ve doğru/yanlış değerler için boolean.
2. **Değer:** Değişkenler, belirli bir değeri tutar. Bu değer, program çalışırken değiştirilebilir.
3. **Kapsam (Scope):** Değişkenlerin nerede kullanılabileceği, tanımlandıkları yere bağlıdır. Bir fonksiyon içinde tanımlanan değişkenler sadece o fonksiyon içinde kullanılabilir.
4. **Yaşam Süresi:** Değişkenler, tanımlandıkları yerde var olur ve bu yerin dışında yok olurlar. Örneğin, bir fonksiyon içinde tanımlanan değişkenler, fonksiyon çalışırken vardır ve fonksiyon bittiğinde bellekten silinir.

Değişken İsimlendirme Kuralları

Değişken isimlendirirken dikkat etmemiz gereken kurallar şunlardır:

1. **Harfler ve Sayılar:** Değişken isimleri harflerle başlamalı ve harfler, sayılar veya alt çizgi (_) içerebilir. Örneğin, sayi1 veya isim gibi.
2. **Özel Karakterler:** Değişken isimlerinde özel karakterler (örneğin, !, @, #) kullanılamaz.
3. **Boşluk:** Değişken isimlerinde boşluk kullanılamaz. Örneğin, ilkDegisken gibi.
4. **Rezerve Kelimeler:** Programlama diline ait özel kelimeler (örneğin, for, while, if) değişken adı olarak kullanılamaz.
5. **Anlamlı İsimler:** Değişken isimlerinin, sakladıkları veri ile anlamlı ve açıklayıcı olması tercih edilir. Örneğin, yas veya notOrtalama gibi.

Değişkenlerin Bellek Üzerindeki Yeri

Değişkenlerin bellek üzerindeki yeri, değişkenin tipine ve kapsamına bağlı olarak değişir:

1. **Stack Belleği:** Yerel değişkenler genellikle stack belleğinde saklanır. Bu bellek alanı hızlı erişim sağlar ve fonksiyon çağrıları sırasında kullanılır.
2. **Heap Belleği:** Dinamik olarak ayrılan ve genellikle nesneler ve büyük veri yapıları için kullanılan bellek alanıdır.
3. **Statik Bellek:** Global değişkenler ve statik değişkenler programın başlangıcında ayrılır ve program süresince aynı bellek alanında kalır.

Örnek Kodlarla Değişken Tanımlama ve Kullanımı

Java dilinde değişken tanımlama ve kullanımı ile ilgili basit örnek:

```
public class Main {
    public static void main(String[] args) {
        // Değişken tanımlama ve değer atama
        int sayi = 10;          // Tam sayı türünde bir değişken
        String isim = "Ali";    // Metin türünde bir değişken
        double pi = 3.14;       // Ondalıklı sayı türünde bir değişken
        boolean dogruMu = true; // Doğru/yanlış türünde bir değişken

        // Değişkenlerin kullanımı
        System.out.println(sayi);      // 10
```

```
        System.out.println(isim);      // Ali
        System.out.println(pi);        // 3.14
        System.out.println(dogruMu);   // true

        // Değişken değerini değiştirme
        sayi = 20;
        System.out.println(sayi);      // 20

        // Farklı tiplerde değişkenler ile işlem yapma
        int yeniSayi = sayi + 5;
        System.out.println(yeniSayi);  // 25

        // Bir fonksiyon içinde yerel değişken tanımlama
        merhabaDe();

        // Global değişken örneği
        GlobalDegiskenler global = new GlobalDegiskenler();
        System.out.println(global.globalSayi);  // 100
        global.globalDegistir();
        System.out.println(global.globalSayi);  // 200
    }

    public static void merhabaDe() {
        String mesaj = "Merhaba, Dünya!";
        System.out.println(mesaj);
    }
}

class GlobalDegiskenler {
    int globalSayi = 100;

    void globalDegistir() {
        globalSayi = 200;
    }
}
```

Bu örneklerde, değişkenlerin nasıl tanımlandığını, değer atandığını ve kullanıldığını görebilirsiniz.

Tam Sayılar (integers)

Tanım: Tam sayılar, negatif veya pozitif, kesirli kısmı olmayan sayılardır. Örneğin, -5, 0, 42 birer tam sayıdır.

Örnek:

```
int yas = 25;  // Tam sayı tipi
int yil = 2024;
```

Ondalık Sayılar (floating point numbers)

Tanım: Ondalık sayılar, kesirli kısmı olan sayılardır. Daha kesin hesaplamalar gerektiren durumlarda kullanılır. Örneğin, 3.14, -0.001, 2.71828.

Örnek:

```
double pi = 3.14;      // Double tipi, ondalıklı sayılar için
float g = 9.81f;       // Float tipi, daha az hassas ondalıklı sayılar için
```

Karakterler ve Dizeler (characters and strings)

Tanım: Karakterler, tek bir harf veya sembolü temsil eder. Dizeler ise birden fazla karakterden oluşan metinlerdir.

Örnek:

```
char harf = 'A';       // Tek bir karakter

String isim = "Ali";   // Birden fazla karakterden oluşan metin
```

Boolean (True/False) Veri Tipi

Tanım: Boolean veri tipi sadece iki değer alabilir: true (doğru) veya false (yanlış). Genellikle koşulların kontrolünde kullanılır.

Örnek:

```
boolean dogruMu = true;  // Doğru/yanlış tipi
boolean gecis = false;
```

Örnek Kodla Temel Veri Tiplerinin Kullanımı

Java dilinde temel veri tiplerinin nasıl kullanıldığını gösteren örnek:

```
public class Main {
   public static void main(String[] args) {
      // Tam Sayılar (integers)
      int sayi = 10;
      System.out.println("Sayı: " + sayi);

      // Ondalık Sayılar (floating point numbers)
      double pi = 3.14;
      System.out.println("Pi: " + pi);

      // Karakterler ve Dizeler (characters and strings)
      char harf = 'A';
      System.out.println("Harf: " + harf);

      String isim = "Ali";
      System.out.println("İsim: " + isim);

      // Boolean (True/False) Veri Tipi
      boolean dogruMu = true;
      System.out.println("Doğru mu: " + dogruMu);

      // Basit Hesaplamalar ve Karşılaştırmalar
      int toplam = sayi + 5;
      System.out.println("Toplam: " + toplam);

      double yaricap = 2.0;
      double alan = pi * yaricap * yaricap;
      System.out.println("Dairenin Alanı: " + alan);

      boolean buyukMu = (sayi > 5);
      System.out.println("Sayı 5'ten büyük mü: " + buyukMu);
   } }
```

Bu kod parçası, temel veri tiplerinin nasıl tanımlandığını ve kullanıldığını gösteren basit örnekler içerir.

Gelişmiş Veri Tipleri

Listeler (arrays/lists)

Tanım: Listeler, aynı tipteki verileri bir arada saklamak için kullanılır. Birden fazla veriyi tek bir değişken altında tutabiliriz. Java'da listeye Array denir.

Örnek:

```
int[] sayilar = {1, 2, 3, 4, 5};  // Tam sayı dizisi
String[] isimler = {"Ali", "Veli", "Ayşe"};  // Metin dizisi
```

Kümeler (sets)

Tanım: Kümeler, benzersiz verileri saklar. Aynı değerden birden fazla bulunamaz. Java'da kümeler Set sınıfı ile kullanılır.

Örnek:

```
import java.util.HashSet;
import java.util.Set;

Set<String> meyveler = new HashSet<>();
meyveler.add("Elma");
meyveler.add("Armut");
meyveler.add("Muz");
```

Sözlükler (dictionaries/maps)

Tanım: Sözlükler, anahtar-değer (key-value) çiftleri şeklinde veri saklar. Her anahtar benzersizdir ve bir değere karşılık gelir. Java'da sözlükler Map sınıfı ile kullanılır.

Örnek:

```
import java.util.HashMap;

import java.util.Map;

Map<String, Integer> yaslar = new HashMap<>();

yaslar.put("Ali", 25);

yaslar.put("Veli", 30);

yaslar.put("Ayşe", 22);
```

Örnek Kodla Gelişmiş Veri Tiplerinin Kullanımı

Java dilinde gelişmiş veri tiplerinin nasıl kullanıldığını gösteren basit bir örnek:

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        // Listeler (arrays/lists)
        int[] sayilar = {1, 2, 3, 4, 5};
        System.out.println("Dizideki ilk sayı: " + sayilar[0]);

        String[] isimler = {"Ali", "Veli", "Ayşe"};
        System.out.println("Dizideki ilk isim: " + isimler[0]);

        // Kümeler (sets)
        Set<String> meyveler = new HashSet<>();
        meyveler.add("Elma");
        meyveler.add("Armut");
        meyveler.add("Muz");
```

```
        System.out.println("Kümedeki meyveler: " + meyveler);

        // Sözlükler (dictionaries/maps)
        Map<String, Integer> yaslar = new HashMap<>();
        yaslar.put("Ali", 25);
        yaslar.put("Veli", 30);
        yaslar.put("Ayşe", 22);
        System.out.println("Ali'nin yaşı: " + yaslar.get("Ali"));

        // Gelişmiş veri tipleri ile basit işlemler
        // Liste içinden eleman çekme
        System.out.println("Dizideki ikinci sayı: " + sayilar[1]);

        // Küme içindeki elemanları yazdırma
        for (String meyve : meyveler) {
            System.out.println("Meyve: " + meyve);
        }

        // Sözlük içinden anahtar ile değer çekme
        int velininYasi = yaslar.get("Veli");
        System.out.println("Veli'nin yaşı: " + velininYasi);
    }   }
```

Bu örneklerde, listeler, kümeler ve sözlüklerin nasıl tanımlandığını ve kullanıldığını görebilirsiniz.

Tip Dönüşümleri

Otomatik Tip Dönüşümleri

Tanım: Otomatik tip dönüşümleri, küçük bir veri tipinin otomatik olarak daha büyük bir veri tipine dönüştürülmesidir. Bu işlem, veri kaybı olmadan gerçekleşir. Örneğin, int türündeki bir değişkeni double türüne dönüştürebiliriz.

```
int sayi = 10;
double buyukSayi = sayi;  // Otomatik dönüşüm, int -> double
System.out.println("Büyük Sayı: " + buyukSayi);  // Çıktı: Büyük Sayı: 10.0
```

Manuel Tip Dönüşümleri

Tanım: Manuel tip dönüşümleri, büyük bir veri tipini daha küçük bir veri tipine dönüştürmektir. Bu işlem sırasında veri kaybı olabilir ve dönüşümü belirtmek için dönüştürmek istediğimiz tipi açıkça belirtmeliyiz.

Örnek:

```
double pi = 3.14;
int tamSayi = (int) pi;  // Manuel dönüşüm, double -> int
System.out.println("Tam Sayı: " + tamSayi);  // Çıktı: Tam Sayı: 3
```

Örnek Kodlarla Tip Dönüşümleri (Java)

Aşağıda, Java dilinde tip dönüşümlerinin nasıl kullanıldığını gösteren basit bir örnek bulunmaktadır:

```
public class Main {

    public static void main(String[] args) {

        // Otomatik Tip Dönüşümleri

        int sayi = 10;

        double buyukSayi = sayi;  // Otomatik dönüşüm, int -> double

        System.out.println("Büyük Sayı: " + buyukSayi);  // Çıktı: Büyük Sayı: 10.0

        // Manuel Tip Dönüşümleri
```

```
        double pi = 3.14;
        int tamSayi = (int) pi;  // Manuel dönüşüm, double -> int
        System.out.println("Tam Sayı: " + tamSayi);  // Çıktı: Tam Sayı: 3

        // String'den Sayıya Dönüşüm
        String metin = "123";
        int sayiMetin = Integer.parseInt(metin);  // String -> int dönüşüm
        System.out.println("Sayı Metin: " + sayiMetin);  // Çıktı: Sayı Metin: 123

        // Sayıdan String'e Dönüşüm
        int sayi2 = 456;
        String metin2 = String.valueOf(sayi2);  // int -> String dönüşüm
        System.out.println("Metin: " + metin2);  // Çıktı: Metin: 456

        // Farklı veri tipleri arasında dönüşüm
        float f = 10.5f;
        double d = f;  // Otomatik dönüşüm, float -> double
        System.out.println("Double: " + d);  // Çıktı: Double: 10.5

        double d2 = 9.99;
        int i2 = (int) d2;  // Manuel dönüşüm, double -> int
        System.out.println("Integer: " + i2);  // Çıktı: Integer: 9
    }   }
```

Bu kod parçası, otomatik ve manuel tip dönüşümlerinin nasıl yapılacağını gösterir.

Sabitler

Sabitler Tanımı ve Kullanımı

Tanım: Sabitler, değerleri program boyunca değişmeyen değişkenlerdir. Java'da sabitler, final anahtar kelimesi ile tanımlanır. Sabitler, bir kez değer atandıktan sonra tekrar değiştirilemezler. Genellikle büyük harflerle isimlendirilirler.

Kullanımı: Sabitler, sabit değerlerin kod içinde tekrarlanmasını önlemek ve bu değerlerin anlamını daha belirgin hale getirmek için kullanılır. Örneğin, matematiksel bir sabit olan PI sayısı veya bir programın maksimum kullanıcı sayısı gibi.

Örnek:

```
final double PI = 3.14159;
final int MAX_KULLANICI_SAYISI = 100;
```

Sabitler ile Değişkenler Arasındaki Farklar

Sabitler:

- ❿ Değeri bir kez atanır ve bir daha değiştirilemez.
- · final anahtar kelimesi ile tanımlanır.
- ❿ Genellikle büyük harflerle isimlendirilir.

Değişkenler:

- ❿ Değeri programın farklı yerlerinde değiştirilebilir.
- · final anahtar kelimesi kullanılmaz.
- ❿ Genellikle küçük harflerle veya camelCase ile isimlendirilir.

Örnek:

```
final int SABIT_DEGER = 50;  // Sabit
int degiskenDeger = 10;  // Değişken
degiskenDeger = 20;  // Değeri değiştirilebilir
// SABIT_DEGER = 60;  // Hata: Sabit değeri değiştirilemez
```

Örnek Kodla Sabitlerin Kullanımı

Java dilinde sabitlerin nasıl kullanıldığını gösteren basit bir örnek:

```
public class Main {
    // Sınıf seviyesinde sabit tanımlama
    public static final double PI = 3.14159;
    public static final int MAX_KULLANICI_SAYISI = 100;

    public static void main(String[] args) {
        // Yerel sabit tanımlama
        final int SABIT_DEGER = 50;

        // Sabitlerin kullanımı
        System.out.println("Pi sayısı: " + PI);
        System.out.println("Maksimum kullanıcı sayısı: " + MAX_KULLANICI_SAYISI);
        System.out.println("Sabit değer: " + SABIT_DEGER);

        // Sabitler ile matematiksel işlemler
        double yaricap = 2.5;
        double alan = PI * yaricap * yaricap;
        System.out.println("Dairenin alanı: " + alan);

        // Değişken tanımlama ve kullanımı
```

```
        int degiskenDeger = 10;
        System.out.println("Değişken değer: " + degiskenDeger);

        // Değişken değeri değiştirme
        degiskenDeger = 20;
        System.out.println("Yeni değişken değer: " + degiskenDeger);

        // Hatalı kullanım: Sabit değeri değiştirmeye çalışmak
        // SABIT_DEGER = 60;  // Hata: Sabit değeri değiştirilemez
    }
}
```

Bu kod parçasında, sabitlerin nasıl tanımlandığını ve kullanıldığını görebilirsiniz.

Veri Tipleri ile İlgili Fonksiyonlar ve Operatörler

Temel Aritmetik Operatörler

Tanım: Aritmetik operatörler, sayısal değerler üzerinde matematiksel işlemler yapmak için kullanılır.

- + (Toplama): İki sayıyı toplar.
- - (Çıkarma): Bir sayıdan diğerini çıkarır.
- * (Çarpma): İki sayıyı çarpar.
- / (Bölme): Bir sayıyı diğerine böler.
- % (Modülüs): Bir sayının diğerine bölümünden kalan değeri verir.

Örnek:

```
int a = 10;
int b = 3;
System.out.println("Toplama: " + (a + b));  // Çıktı: Toplama: 13
System.out.println("Çıkarma: " + (a - b));  // Çıktı: Çıkarma: 7
System.out.println("Çarpma: " + (a * b));   // Çıktı: Çarpma: 30
System.out.println("Bölme: " + (a / b));    // Çıktı: Bölme: 3
System.out.println("Modülüs: " + (a % b));  // Çıktı: Modülüs: 1
```

Karşılaştırma Operatörleri

Tanım: Karşılaştırma operatörleri, iki değeri karşılaştırmak için kullanılır ve sonuç olarak true veya false döner.

- == (Eşittir): İki değerin eşit olup olmadığını kontrol eder.

- != (Eşit Değildir): İki değerin eşit olup olmadığını kontrol eder.
- > (Büyüktür): Sol değerin sağ değerden büyük olup olmadığını kontrol eder.
- < (Küçüktür): Sol değerin sağ değerden küçük olup olmadığını kontrol eder.
- >= (Büyük veya Eşittir): Sol değerin sağ değerden büyük veya eşit olup olmadığını kontrol eder.
- <= (Küçük veya Eşittir): Sol değerin sağ değerden küçük veya eşit olup olmadığını kontrol eder.

Örnek:

```
int x = 5;

int y = 10;

System.out.println("Eşit mi: " + (x == y));       // Çıktı: Eşit mi: false

System.out.println("Eşit değil mi: " + (x != y)); // Çıktı: Eşit değil mi: true

System.out.println("Büyük mü: " + (x > y));       // Çıktı: Büyük mü: false

System.out.println("Küçük mü: " + (x < y));       // Çıktı: Küçük mü: true

System.out.println("Büyük veya eşit mi: " + (x >= y)); // Çıktı: Büyük veya eşit mi: false

System.out.println("Küçük veya eşit mi: " + (x <= y)); // Çıktı: Küçük veya eşit mi: true
```

Mantıksal Operatörler

Tanım: Mantıksal operatörler, boolean değerlerle çalışır ve mantıksal işlemler yapar.

- && (Ve): Her iki ifade de true ise true döner.
- || (Veya): İki ifadeden biri true ise true döner.
- ! (Değil): İfadenin tersini döner (true ise false, false ise true).

Örnek:

```
boolean a = true;

boolean b = false;

System.out.println("Ve operatörü: " + (a && b));   // Çıktı: Ve operatörü: false

System.out.println("Veya operatörü: " + (a || b)); // Çıktı: Veya operatörü: true

System.out.println("Değil operatörü: " + (!a));    // Çıktı: Değil operatörü: false
```

Veri Tipleriyle İlgili Yerleşik Fonksiyonlar

Tanım: Java'da yerleşik olarak gelen bazı fonksiyonlar, veri tipleri ile ilgili işlemler yapmamıza olanak tanır. Örneğin, sayıları metne veya metinleri sayıya dönüştürmek için kullanılır.

Örnek:

```
// Sayıyı metne dönüştürme

int sayi = 123;

String sayiMetin = String.valueOf(sayi);

System.out.println("Sayı Metin: " + sayiMetin); // Çıktı: Sayı Metin: 123


// Metni sayıya dönüştürme
```

```
String metin = "456";
int metinSayi = Integer.parseInt(metin);
System.out.println("Metin Sayı: " + metinSayi); // Çıktı: Metin Sayı: 456
```

Örnek Kodlarla Fonksiyonlar ve Operatörler (Java)

Aşağıda, Java dilinde fonksiyonlar ve operatörlerin nasıl kullanıldığını gösteren basit bir örnek bulunmaktadır:

```
public class Main {
    public static void main(String[] args) {
        // Temel Aritmetik Operatörler
        int a = 10;
        int b = 3;
        System.out.println("Toplama: " + (a + b));  // Çıktı: Toplama: 13
        System.out.println("Çıkarma: " + (a - b));  // Çıktı: Çıkarma: 7
        System.out.println("Çarpma: " + (a * b));   // Çıktı: Çarpma: 30
        System.out.println("Bölme: " + (a / b));    // Çıktı: Bölme: 3
        System.out.println("Modülüs: " + (a % b));  // Çıktı: Modülüs: 1

        // Karşılaştırma Operatörleri
        int x = 5;
        int y = 10;
        System.out.println("Eşit mi: " + (x == y));       // Çıktı: Eşit mi: false
        System.out.println("Eşit değil mi: " + (x != y)); // Çıktı: Eşit değil mi: true
        System.out.println("Büyük mü: " + (x > y));       // Çıktı: Büyük mü: false
        System.out.println("Küçük mü: " + (x < y));       // Çıktı: Küçük mü: true
        System.out.println("Büyük veya eşit mi: " + (x >= y)); // Çıktı: Büyük veya eşit mi: false
        System.out.println("Küçük veya eşit mi: " + (x <= y)); // Çıktı: Küçük veya eşit mi: true

        // Mantıksal Operatörler
        boolean p = true;
        boolean q = false;
        System.out.println("Ve operatörü: " + (p && q));   // Çıktı: Ve operatörü: false
        System.out.println("Veya operatörü: " + (p || q)); // Çıktı: Veya operatörü: true
        System.out.println("Değil operatörü: " + (!p));    // Çıktı: Değil operatörü: false

        // Veri Tipleriyle İlgili Yerleşik Fonksiyonlar
        int num = 123;
        String numStr = String.valueOf(num); // Sayıyı metne dönüştürme
```

```
        System.out.println("Sayı Metin: " + numStr); // Çıktı: Sayı Metin: 123


        String str = "456";
        int strNum = Integer.parseInt(str); // Metni sayıya dönüştürme
        System.out.println("Metin Sayı: " + strNum); // Çıktı: Metin Sayı: 456
    }
}
```

Bu örneklerde, temel aritmetik operatörler, karşılaştırma operatörleri, mantıksal operatörler ve veri tipleriyle ilgili yerleşik fonksiyonların nasıl kullanıldığını görebilirsiniz.

Hatalar ve Sorun Giderme

Veri Tipi Hataları ve Nedenleri

Tanım: Veri tipi hataları, bir değişkene yanlış veri tipi atamaktan veya beklenmeyen veri tipleriyle işlemler yapmaktan kaynaklanır. Bu hatalar, programın derlenmemesine veya çalışırken çökmesine neden olabilir.

Nedenleri:

- ⓾ Yanlış veri tipi kullanımı (örneğin, bir metni sayıya dönüştürmeye çalışmak).
- ⓾ Otomatik tip dönüşümünün yanlış anlaşılması.
- ⓾ Dizilerde (arrays) veya koleksiyonlarda (collections) yanlış veri tipi kullanımı.
- ⓾ Değişkenin türü ile uyumsuz değer atamaları.

Örnek:

```
String metin = "123";
int sayi = metin; // Hata: String türünden bir değişkeni int türüne atamaya çalışmak
```

Hataları Bulma ve Çözme Yöntemleri

Yöntemler:

1. **Derleyici Hataları:** Derleyici (compiler) tarafından raporlanan hataları dikkatlice okuyun. Genellikle hata türü ve hatanın bulunduğu satır hakkında bilgi verir.
2. **Hata Mesajları:** Hata mesajlarını dikkatlice inceleyin. Bu mesajlar, hatanın sebebini ve nasıl düzeltileceğini anlamanızı sağlar.
3. **Kod İncelemesi:** Kodunuzu dikkatlice gözden geçirin ve değişkenlerin veri tiplerini kontrol edin. Hangi satırın hataya neden olduğunu anlamaya çalışın.
4. **Loglama (Logging):** Programınıza loglama ekleyerek, belirli noktaların çalışıp çalışmadığını kontrol edebilirsiniz. Bu, özellikle karmaşık kodlarda faydalıdır.
5. **Debugger Kullanımı:** Debugger araçları kullanarak kodunuzu adım adım çalıştırabilir ve değişken değerlerini izleyebilirsiniz. Bu, hataları bulmanıza ve anlamanıza yardımcı olur.

Örnek Kodlarla Hata Giderme (Java)

Java dilinde yaygın veri tipi hataları ve bu hataların nasıl giderileceğine dair örnekler:

Örnek 1: Yanlış Veri Tipi Ataması

Yanlış Kod:

```
public class Main {
    public static void main(String[] args) {
        String metin = "123";
        int sayi = metin; // Hata: String türünden bir değişkeni int türüne atamaya çalışmak
        System.out.println("Sayı: " + sayi);
    }
}
```

Düzeltme:

```
public class Main {
    public static void main(String[] args) {
        String metin = "123";
        int sayi = Integer.parseInt(metin); // Doğru: String'i int'e dönüştürmek için parseInt kullanın
        System.out.println("Sayı: " + sayi);
    }
}
```

Örnek 2: NullPointerException Hatası

Yanlış Kod:

```
public class Main {
    public static void main(String[] args) {
        String metin = null;
        int uzunluk = metin.length(); // Hata: null değeri üzerinde length() çağrısı yapmak
        System.out.println("Uzunluk: " + uzunluk);
    }   }
```

Düzeltme:

```
public class Main {
    public static void main(String[] args) {
        String metin = null;
        if (metin != null) { // Null kontrolü eklemek
            int uzunluk = metin.length();
            System.out.println("Uzunluk: " + uzunluk);
        } else {
            System.out.println("Metin değeri null");
        }    }    }
```

Küçük Bir Hesap Makinesi Uygulaması

Projede Kullanılacak Değişkenler ve Veri Tipleri

Bu hesap makinesi uygulamasında kullanılacak temel değişkenler ve veri tipleri şunlardır:

- double türünde sayılar için değişkenler: İşlemlerde kullanılacak olan sayıları saklamak için.
- char türünde işlem seçenekleri için değişken: Kullanıcıdan hangi işlemi yapmak istediğini seçmesi için.
- Scanner nesnesi: Kullanıcıdan giriş almak için kullanılacak.

Projenin Kodu ve Açıklamaları

```
import java.util.Scanner;


public class HesapMakinesi {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);


        // Kullanıcıya işlem seçeneklerini göster
        System.out.println("Hesap Makinesine Hoş Geldiniz!");
        System.out.println("Yapmak istediğiniz işlemi seçin:");
```

```
System.out.println("1. Toplama (+)");
System.out.println("2. Çıkarma (-)");
System.out.println("3. Çarpma (*)");
System.out.println("4. Bölme (/)");

// Kullanıcıdan işlem seçeneğini al
System.out.print("Seçeneği girin (1/2/3/4): ");
char secenek = scanner.next().charAt(0);

// İki sayıyı kullanıcıdan al
System.out.print("İlk sayıyı girin: ");
double sayi1 = scanner.nextDouble();
System.out.print("İkinci sayıyı girin: ");
double sayi2 = scanner.nextDouble();

double sonuc = 0;

// İşlem seçeneğine göre hesaplama yap
switch (secenek) {
    case '1':
        sonuc = sayi1 + sayi2;
        break;
    case '2':
        sonuc = sayi1 - sayi2;
        break;
    case '3':
        sonuc = sayi1 * sayi2;
        break;
    case '4':
        // Bölme işlemi yaparken sıfıra bölme hatasını kontrol et
        if (sayi2 != 0) {
            sonuc = sayi1 / sayi2;
        } else {
            System.out.println("Hata: Sıfıra bölme hatası!");
            return;
        }
```

```
                break;
            default:
                System.out.println("Geçersiz seçenek!");
                return;
        }

        // Sonucu ekrana yazdır
        System.out.println("Sonuç: " + sonuc);

        // Scanner kapat
        scanner.close();
    }
}
```

Bu kod parçası, kullanıcıya basit bir hesap makinesi uygulaması sunar. Kullanıcı, toplama, çıkarma, çarpma veya bölme işlemlerinden birini seçebilir ve ardından iki sayı girebilir.