Java büyük küçük harf duyarlı bir dil.

Dışarıya bir çıktı vermek istersek System.out kullanıyoruz

Variables

Bir değişken atayıp birden fazla yerde kullanabiliyoruz. Değişkenin değeri değiştirildiği zaman tüm kullanıldığı yerlerde değişiyor.

Önce veri tipini sonra ismini sonra değerini giriyoruz.

int ogrenciSayisi = 12;

Metin türünde bir değişken tanımlayalım.

String mesaj= “öğrenci sayisi:”;

Bu ikisini birlikte yazdırmak için

System.out.println(mesaj+ogrenciSayisi);

Ondalıklı sayı tanımlarken double kullanıyoruz

Genelde int ve double kullanılır.

char tek bir karakter tanımlamak için kullanılır.

boolean doğru ya da yanlış 1/0. Çok yoğun kullanılır.

Şart blokları(if/else/else if)

Şart gerçekleştiğinde çalışacak kod bloklarını if else kullanarak yapıyoruz.

int sayi = 15;

if(sayi<15){

System.out.println(“Sayı 15 tan küçüktür”);

}else if(sayi ==15){

System.out.println(“Sayı 15 e eşittir.”);

}else {

System.out.println(“Sayı 15 ten büyüktür”);

}

İf else kullanarak en büyük sayıyı bulma

int sayi1 = 10;  
 int sayi2 = 20;  
 int sayi3 = 15;  
 int enBuyuk=sayi1;  
 if (enBuyuk<sayi2){  
 enBuyuk=sayi2;  
 }  
 if (enBuyuk<sayi3){  
 enBuyuk=sayi3;  
 }  
 System.*out*.println("En buyuk sayı: "+enBuyuk);  
 }  
}

Switch

Fazla kullanılmaz genelde if else kullanılır. Kullancağımız değişken ismini şart kısmına yazıyoruz. Case kullanarak koşulları ve ne olacağını belirtiyoruz. Break ile koşulları bitiriyoruz. Farklı değerde ne olucağını belirtmek için default kullanıyoruz.

char grade = 'A';  
switch (grade){  
 case 'A':  
 System.*out*.println("A");  
 break;  
 case 'B':  
 System.*out*.println("B");  
 break;  
 case'C':  
 System.*out*.println("C");  
 break;  
 case'D':  
 System.*out*.println("D");  
 break;  
 case 'F':  
 System.*out*.println("F");  
 break;  
 default:  
 System.*out*.println("Yanlış değer girindiniz");  
}

Döngüler

For

Döngü oluştururken kullanıyoruz. Bir koşulun başka bir koşul olana kadar devam etmesi diyebiliriz.

Sayaç Örneği

For (int i=1;i<10;i++){

System.out.println(i);

}

System.out.println(“Döngü bitti”);

While

Yukarıda yaptığımız işlemin aynısını while kullanarak yapalım.

int i=1;  
while (i<10){  
 System.*out*.println(i);  
 i++;  
}  
System.*out*.println("Döngü bitti");

Do While

Çok nadir kullanılır. Döngü şartı sağlanmasa bile bir kere döngüye girer. Ekrana sadece 100 yazdırıp döngüyü bitirir. Genelde koşul gerçekleşmese bile log kaydı atmak için kullanılır.

int j=100;  
do{  
 System.*out*.println(j);  
 j++;  
}while (j<10);  
System.*out*.println("do-while döngüsü bitti");

Arrays

Dizileri oluşturmak için önce tipini belirtiyoruz yanına dizi anlamına gelen [] işaretini koyuyoruz. İsim veriyoruz. Eleman sayısını belirtiyoruz.

String[] ogrenciler = new String[3];  
ogrenciler[0] = "Yunus Emre";  
ogrenciler[1] = "Mehmedor";  
ogrenciler[2] = "Kader";  
  
for (String ogrenci: ogrenciler){  
 System.*out*.println(ogrenci);  
}

Dizileri şu şekilde de oluşturabiliyoruz;

double[] numaralar = {1.6, 2.3, 4.6, 6.7, 9.0};  
double toplam = 0;  
double enBuyuk = numaralar[0];  
for (double numara : numaralar) {  
 if (enBuyuk < numara) {  
 enBuyuk = numara;  
 }   
 toplam = toplam + numara;  
 System.*out*.println(numara);  
}  
System.*out*.println("Numaralar toplamı: "+toplam);  
System.*out*.println("En buyuk sayı: "+enBuyuk);

Strings(Metinler)

String dizisindeki eleman sayısını bulmak için;

System.out.println(“Elaman sayısı: ”+mesaj.length());

Metin içerisindeki karakteri bulma;

System.out.println(“5.eleman : “+mesaj.charAt(4));

Metinleri birleştirerek yeni bir metin oluşturuyor. Concatenation

System.out.println(mesaj.concat(“Yaşasın”);

Bununla başlıyor/bitiyor mu? Bize boolean bir sonuç verir.

System.out.println(mesaj.starswith(“B”));

System.out.println(mesaj.endswith(“.”);

Karakterleri toplu şekilde getirmek için;

char[] karakterler = new char[5]

mesaj.getchars(0,5,karakterler,0);

System.out.println(karakterler);

Çıktı:0 dahil 5 e kadar olan karakterleri getirir.

Girdiğimiz karakterin kaçıncı indexte olduğunu öğrenme

System.out.println(mesaj.ındexOf(“n”)); //Soldan sağa bakar ılk bulduğunu yazdırır

System.out.println(mesaj.lastIndexOf(“a”)); // Sağa en yakın olanı yazdırır en sndakini;

Metindeki karakterleri değiştirme işlemi;

String yeniMesaj = mesaj.replace(‘ ’,’-’); //çıktı: Boşlukla – yı degıstırır.

Metin içerisinden girdiğimiz indeksler arasındaki karakterleri getirme;

System.out.println(measj.substring(2,5));

Metindeki gelimeleri bölmek için (Boşluktan bölüyoruz);

For (String kelime : mesaj.split(“ ”){

System.out.println(kelime);

}

Metindeki tüm harfleri küçültmek/büyültmek için;

System.out.println(mesaj.toLowerCase();

System.out.println(mesaj.toUpperCase();

Metin başındaki ve sonundaki boşlukları silmek;

System.out.println(mesaj.trim());

Alıştırmalar

Asal sayı bulma örneği;

int sayi = 1;  
boolean asalSorgu= true;  
  
if (sayi==1){  
 System.*out*.println("Sayı asal değildir");  
 return;  
}  
if (sayi<1){  
 System.*out*.println("Geçersiz sayı girdiniz");  
 return;  
}  
  
for(int i=2; i<sayi ;i++){  
 if(sayi % i ==0){  
 asalSorgu=false;  
 }  
}  
if (asalSorgu==true){  
 System.*out*.println("Sayı Asaldır");  
}else {  
 System.*out*.println("Sayı Asal Değildir.");  
}

Kalın ince harf kontrol

char grade = 'A';  
switch (grade){  
 case 'A':  
 case 'B':  
 case 'C':  
 case'D':  
 case 'F':  
 System.*out*.println("Kalın sesli harf");  
 break;  
 default:  
 System.*out*.println("İnce sesli harf");

Mükemmel sayı bulucu

int number = 28;  
int total = 0;  
  
for (int i =1; i<number ;i++){  
 if (number % i == 0){  
 total= total+i;  
 }  
} if (total == number){  
 System.*out*.println("Mükemmel sayı");  
 }else {  
 System.*out*.println("Mükemmel sayı değil");  
 }

Sayı bulma

int[] sayilar = new int[]{1,2,5,7,9,0};  
int aranacak = 5;  
boolean varMi = false;  
  
for (int sayi:sayilar){  
 if(sayi==aranacak){  
 varMi= true;  
 }  
}  
if (varMi){  
 System.*out*.println("Sayı mevcuttur.");  
}else {  
 System.*out*.println("Sayı mevcut değildir.");  
}

Methodlarla çalışmak

Bir fonksiyonu/ methodu bir kere yazıp sonradan kullanabiliyoruz farklı sayfalarda ya da kod bloklarında tekrar yazmamıza gerek kalmıyor.

Public static void main(String[]args){

İnt sayi = topla(15,7);

}

Public static int topla(int sayi1, int sayi 2){

return sayi1+sayi2;

}

Void olduğu zaman bir şey döndürmüyor. O yüzden kullanıcağımız veri tipini giriyoruz. Bu şekilde method oluşturduğumuz zaman kesin bir şey döndürmemiz gerekiyor(return)

Variable arguments

Yukarıdaki örnekte argümanları 2 olarak sınırlandırdık. Sınırsız argümanla çalışmak için kullanılır.

Public static void main(String[]args){

int sayi = topla(10,5,5,7);

System.out.println(sayi) --🡪 çıktı 27

Public static int topla(int… sayilar){

İnt toplam = 0;

For (int sayi: sayilar){

toplam+=sayi;

}

Return toplam;

}

Classes (Sınıflar) (Nesne yönelimli)

Class açmak için : src->new->class

Classlar çok önemlidir

Classlar Reference type’tır. Reference type nasıl çalışır;

CustomerManager customerManager = new CustomerManager();  
CustomerManager customerManager2 = new CustomerManager();  
customerManager=customerManager2;

Bu kod yazıldığı zaman

101 101

{ }

CM

CM2

102 102

{ }

customerManager=customerManager2 BU KOD YAZILDIĞI İÇİN CMNIN REFERANS NUMARASI 101 DEN 102 OLUYOR VE KARŞI TARAFTA OLUŞTURULAN NESNE 101 REFERANS NUMARALI NESNE KİMSEYLE EŞLEŞMEDİĞİ İÇİN ÇÖP OLUYOR.

New pahalı bir iştir bu sebeple böyle çöp olacak nesneler oluşturmamak gerekir.

Dizilerde reference type çalışır.

Value type ise şu şekilde çalışır

int sayi1 = 10;  
int sayi2 = 20;  
sayi2=sayi1;  
sayi1 = 30;  
System.*out*.println(sayi2);

Çıktı-> 10

101

Sayi 1 = ~~10~~ 30

Burayla işi yok

Sayi 2 = ~~20~~ 10

102

Main dışında class oluşturup mainde çağırma

Öncelikle src kısmına sağ tıklayıp yeni bir class oluşturuyoruz.

Classımızın ismine DortIslem ismini veriyoruz. İçerisine kodları yazıyoruz.

public class DortIslem {  
 public int Topla(int sayi1, int sayi2){  
 return sayi1+sayi2;  
 }  
 public int Cikar(int sayi1, int sayi2){  
 return sayi1-sayi2;  
 }  
 public int Carp(int sayi1, int sayi2){  
 return sayi1\*sayi2;  
 }  
 public int Bol(int sayi1, int sayi2){  
 return sayi1/sayi2;  
 }  
}

Kodları yazdıktan sonra main kısmına gidip classı çağırıyoruz.

public class Main {  
  
 public static void main(String[] args) {  
 DortIslem dortIslem = new DortIslem();  
 int sonuc = dortIslem.Carp(5,5);  
 System.*out*.println(sonuc);  
  
 }  
}

Single responsibility principle (Tek sorumluluk ılkesı)

Bir class bir şeyi yapsın karışık olmasın demek

Önce ürünler diye bir class acıyoruz.

Public class product {

//attribute-özellik

int id;

String name;

String description;

double price;

int stockAmount;

}

Yani bu clasta sadece ürünlerin özelliklerini tanımlıyoruz fakat ürünleri yönetmek için başka bir class açıyoruz.

Public class Product Manager{

Public void Add(Product product){

//JDBC ile ileride veri tabanına yazı yazılacak

System.out.println(“ürün eklendi”+product.name);

}

}

Main kısmına gidip ürün niteliklerini giriyoruz ileride kullanıcıdan alıcaz.

Product product = new Product();

product.name=”Laptop”;

product.id=1;

product.description=”Asus Laptop”;

product.price=5000;

product.stockAmount=3;

ProductManager productmanager = new ProductManager();

product.manager.Add(product);

}

Getter/Setter

Get okumaya set yazmaya değiştirmeye yarar.

Private ile tanımlanan bir değişken sadece tanımlandığı classta cagırılabılır. Diğer classlardan erısılmesını ıstıyorsak get/set kullanıyoruz.

Private int id;

Public int getId();

return id;

}

Public void SetId(int id){ //Set edilen değer parametre olarak verilmeli

This.id=id

}

//This bu classta demek

Mainde bir değişiklik yapacağımız zaman

Product.setId(1);

Şeklinde yapıyoruz.

Method Overloading

Aşırı yükleme anlamına geliyor bir classta iki farklı method gıbı düşünebiliriz.

public class DortIslem {  
  
 public int topla(int sayi1, int sayi2){  
 return sayi1+sayi2;  
 }  
 public int topla(int sayi1, int sayi2, int sayi3){  
 return sayi1+sayi2+sayi3;  
 }  
}

Burada iki şekildede program hata vermeden çalışıyor yani 2 sayı yazılsada 3 sayı yazılsada toplama ıslemı yapılıyor.

public class Main {  
  
 public static void main(String[] args) {  
  
 DortIslem dortIslem = new DortIslem();  
 System.*out*.println(dortIslem.topla(2,3,4));  
  
 }  
}

inheritance(Miras)(extends)

2 tane classımız var biri customer diğeri employee ikisindede id,firstname,lastname, age var. Bunları ikisindede tanımlamak yerine başka bir class’a tanımlayıp extends kodu yazarak miras çekiyoruz.

Person isimli bir class açıp ortak olanları buraya yazdık. Customerdan çekip farklı olan değişkeni tanımlıyoruz.

public class Customer extends Person{

String email;

}

Bir class sadece bir classı extend edebilir.

Spagetti(karışık) iç içe iflerle programlama yapmak yanlış o yüzden reference type olan inheritanceyi çok kullanırız.

Bir bankacılık uygulaması düşünelim. İlk olarak temel kredi hesaplama algoritmasını BaseKrediManager için yazıyoruz.

public class BaseKrediManager {  
 public void Hesapla(){  
 System.*out*.println("Kredi hesaplandı");  
 }

Daha sonra kredi sınıflarında Tarım,Öğretmen,asker bu base classımızı extends ediyoruz.

public class OgretmenKrediManager extends BaseKrediManager{  
}

Hepsinde bu işlemi uyguladıktan sonra KrediUI yani ekranda görünecek şeyi tanımlıyoruz.

ublic class KrediUI {  
 public void KrediHesapla(BaseKrediManager baseKrediManager){  
 baseKrediManager.Hesapla();  
 }

Daha sonra mainde bu KrediUI çağırıyoruz.

public static void main(String[] args) {  
KrediUI krediUI = new KrediUI();  
krediUI.KrediHesapla(new AskerKrediManager());  
 }

Polymorphism(Çok biçimlilik)

Inheritance olmadan kullanılamaz

Overriding yapabiliriz

Bağımlı değil bağımsız sistemler yazmamız gerekiyor.

Class miras aldığı classı ezebilir aynı imzayı kullanarak.

Final keywordu kullanılırsa ezemez

Overriding örneği Şokomelli

İlk olarak miras vereceğimiz baseyi oluşturuyoruz.

public class BaseKrediManager {  
 public double hesapla(double tutar){  
 return tutar \* 1.18;  
 }

TarimKredisi,OgretmenKredisi,OgrenciKredisi classlarını oluşturup BaseKrediManageri extends ediyoruz. Fakat öğrenci kredisi kısmında overriding yapıyoruz.

public double hesapla(double tutar){  
 return tutar \* 1.10;  
}

Birebir aynısını yazıp değiştirerek eziyoruz.

Main kısmında bunları çağırmak için BaseKrediManager dizisi oluşturuyorum.

BaseKrediManager[] krediManagers = new BaseKrediManager[]{new OgretmenKrediManager(), new TarimKrediManager(), new OgrenciKrediManager()};  
for (BaseKrediManager krediManager: krediManagers){  
 System.*out*.println(krediManager.hesapla(1000));  
 }

Çıktı ->1180.0

1180.0

1100.0

Abstract class

Diğer sınıflarla aynıdır fakat base sınıfımızda abstract ettiğimiz kısım diğer tüm sınıflarda tanımlanmak zorundadır. Direkt miras alınamaz. Yani basede şöyle bir şey yazdığımız zaman;

Public abstract class GameCalculator{

Public abstract void hesapla();

}

Bunu yazdığımız zaman extends ettiğimiz classlarda

@ovveride

public void hesapla(){

}

Yazmamız gerekir.

Örnek;

Farklı databaselerle çalışan bir programımız olduğunu düşünelim. Farklı databaseler farklı kodlarla çalışacağı için bunun belirtilmesi gerekir.

İlk olarak miras bırakacak classımızı Basemizi abstract olarak yazıyoruz.

public abstract class BaseDatabaseManager {  
 public abstract void getData();  
}

Daha sonra diğer classlarımızı oluşturup bunu extends edıyoruz.

public class OracleDatabaseManager extends BaseDatabaseManager{  
 @Override  
 public void getData() {  
 System.*out*.println("Veri getirildi: Oracle");  
 }

Diğer classlar içinde aynısını yapıyoruz. Sonra bunu kullanacağımız customer manager classına gidiyoruz.

public class CustomerManager {  
 BaseDatabaseManager databaseManager;  
  
 public void getCustomers(){  
 databaseManager.getData();  
 }

Daha sonra mainde çağırmak için

CustomerManager customerManager = new CustomerManager();  
customerManager.databaseManager = new OracleDatabaseManager();  
customerManager.getCustomers();  
 }

Interface

Src-new-ınterface

İnterfacenin üyeleri direkt olarak publictir. Bu yüzden void Add(); diye yazarız.

Abstractlara çok benzer ama bir class birden fazla ınterfaceyi implement edebilir.

Abstractlar gibi bu da newlenmıyor.

**Örnek**

public interface ICustomerDal{

void add();

}

Classımızda implement etmek için

Public class OracleCustomerManager implements ICustomerDal{

@Override

Public void Add();

}

Mainde kullanmak için

ICustomerDal customerDal = new OracleCustomerManager();

//CustomerDal oraclecustomermanagerın referansını tutuyor.

Interface demo

Bir lojistik firmasını düşünelim hem kendi çalışanları var hem de taşeron işçi alıyor. Kendi çalışanlarımızın yemek maaşını veriyoruz fakat taşeron işçilerin kendi firması karşılıyor. Hepsinde çalışan ortak fakat n kendi çalışanlarımızın fazladan fonksiyonları var. Bunun için Interface kullanıyoruz.

İlk olarak work ınterface açıyoruz.

Public interface IWorkable{

Void work();

}

Daha sonra yemek için interface oluşturuyoruz.

Public interface IEatable{

Void Eat();

}

Maaş için;

Public interface IPayable{

Void pay();

İnterfacelerimizi oluşturduk şimdi kendi çalışanlarımız için worker classı oluşturup workable,Eatable,payable interfacelerini implement ediyorum.

Public class worker implements Iworkable,IEatable,IPayable{

@override

Public void work(){

}

@override

Public void eat(){

}

@override

Public void pay(){

}

}

//Taşeron işçiler için sadece workable implement ediyoruz

Composition-Inner class

! Değilse operetörü

Static yaparsak bir kere oluşur ve herkes kullanır. Hesaplama gibi araçlarda static kullanıyoruz.

Static yaptığımız zaman . koyunca fonksiyona erişilir.

Newlediğimiz şey kullanmadığımızda hafızadan silinir ama static uygulama yenilenene kadar silinmez bellekte kalır.

Manager sınıfları asla static yapılmaz çünkü ilk kullanıcı.

Constractorlarda newleyince çalışır.

Static ana classta kullanılmaz ama inner classta kullanılır.

Packages

Paket eklemek için üst kısma import ediyoruz.

İmport java.util.scanner;

Bu yazdığımız kod konsoldan bize girdi aldırıyor.

Kullanıcağımız paketleri bu şekilde import ediyoruz.

Kendi paketimizi oluşturmak için src->new->packages

Paket isimleri küçük harfle yazılır.

Kendimize matematik isimli package oluşturup içine classları yerleştirdik.

Mainde tek tek import etmek yerine

İmpport matematik.\*; yazabiliriz.

Matematik paketindeki tüm classları import eder.

ArrayList

Diziler gibi ama sonradan eleman eklenebilirler.

Alt enter ile import ediyoruz kullanmak için

Temel arraylist oluşturma->

Arraylist sayilar = new ArrayList();

Sayilar.add(1);

Sayilar.add(10);

Sayilar.add(“Ankara”);

Sayilar.size();->Arraylistte kaç eleman olduğunu gösterir.

Sayilar.set(2,100);-> ikinci elemanı 100 ile değiştirir.

Sayilar.get(2)->ikinci elemanı getirir.

Sayilar.remove(0);->0. İndexteki elemanı siler.

Sayilar.clear();-> Tüm elemanı siler.

Elemanları gezmek için

For (object i:sayilar){

System.out.println(i)

}

Bu şekilde tipini belirlemeden ArrayList oluşturduğumuz zaman içindekiler obje olarak kabul edilir ve her türden veri girebiliriz. Eğer karışık bir veri setiyle çalışmıyorsak Type-Safe olarak ArrayList tanımlarız.

ArrayList<String> sehirler = new ArrayList<Sehirler>();

Sehirler.add(“Ankara”);

Sehirler.add(“İzmit”);

Sehirler.add(“Bilecik”);

//Şehirleri alfabetik sıraya sokmak için

Collections.sort(sehirler);

For(String sehir:sehirler){

System.out.println(sehir);

}

Çıktı -> Ankara

Bilecik

İzmit

Hata yönetimi

Try-Catch

Kodu Try-catch arasına yazıyoruz. Eğer kod Çalışırsa Try çalışmazsa catch bloğu çalışıyor. Sonda her zaman finally çalışıyor. Bazen veri tabanıyla iletişim kurmak gerekir. Eğer bu iletişim uygulama çökerse ve kapatılmazsa sıkıntılar meydana gelir. Finally bloğunda bu iletişim kapatılır.

try {  
 int[] sayilar = new int[]{1, 2, 3};  
 System.*out*.println(sayilar[2]);  
} catch (Exception exception) {  
 System.*out*.println(exception);  
} finally {  
 System.*out*.println("Ben her türlü çalışırım");  
}

**Exception Hiyerarşisi**

**Unchecked exception:** Java kontrol etmıyor Try Catch içine yazmalıyız.

Exception bizim kodlarla yazarken yaptığımız hatadır.

Error bizim kodlarla müdahale edemediğimiz olaylardır.

IOException: Bir dosyaya ulaşamadığımız zaman alınan hatadır(kodsal)

ArithmeticException:Sayısal hatalar(Sisteme girmek için yaş sınırı var o yaş sınırının altında)

NullPointerException: Referansı olmayan bir classı kullanmaya çalışınca alınır.

ClassCastException: Type dönüşüm hatası(Bibrine dönüştürülemeyecek şeyleri dönüştürmeye çalışma)

ArrayIndexoutofBoundsExcepion: Dizi sınırları dışında

Birden fazla catch yapabiliriz. Bu hata olursa buraya yönlendir diğer durumlarda loğla ben bakıyım tarzında.

Checked Exception örnek

Önce src tıklayıp file diyoruz. Bir text dosyası oluşturuyoruz. Sayilar.txt isminde

Mainde bu dosyayı okumak için

BufferedReader reader = BufferedReader(new fileReader (“Dosya dizini”));

IOExceptionlar try catch içine yazılmak zorundadır.(Dosya okumalar)

**Finally bloğunda reader kapatılmalıdır.**

2 yöntem var bunun ıcın finally bloğuna try catch yazmak.

Finally{

Try

{

Reader.(loser);

}catch(Exception exception){

}

Ya da En başa throw IOException yazmak

Public static void main(String[]args) throws IOException{

Text dosyasındaki sayıları satır satır okuyor. İntegera çevirmek için

Total+=Integer.valuof(line);->integera çevirir.

public static void main(String[] args) {  
 BufferedReader reader = null;  
 int total=0;  
 try {  
 reader = new BufferedReader(new FileReader("C:\\javademos\\readingFileDemo\\src\\sayilar.txt"));  
 *//Satır satır okundugu ıcın line dıyoruz Satır bosta olabılır oyuzden null dıyoruz. .* String line =null;  
 *//Eğer okunabılıyorsa bunu yap* while ((line = reader.readLine())!=null){  
 total+=Integer.*valueOf*(line);  
 }  
 System.*out*.println("Toplam = " +total);  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 try {  
 reader.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }

Bazen diğer sayfalara throw göndeririz çünkü onu kullanacak kişi try catch olarak yazsın yazmadığı sürece hata versin isteriz. Bu örnekte classta sadece if else yaparak bakıye dusuk olma durumunuda kontrol edebılırdık ama throwu görmek ıcın yaptık. Ayrıca mainde signature olarak yukarıya throw Exception yazsakta olurdu ama bız ana sayfada try Catch ıcıne yazdık Önce classı ıncelıycez.

public class HesapYonetimi {  
 private double hesap;  
  
  
 public void yatirma (double tutar){  
 hesap = getHesap() + tutar;  
 }  
  
 public void cekme (double tutar) throws Exception {  
 if (hesap>=tutar){  
 hesap = getHesap() - tutar;  
 }else {  
 throw new Exception("Bakiye Yetersiz");  
 }  
  
 }  
 public double getHesap() {  
 return hesap;  
 }  
}

**Main kısmını inceleyelim**

public class Main {  
  
 public static void main(String[] args) {  
 HesapYonetimi hesapYonetimi = new HesapYonetimi();  
 System.*out*.println("Hesap Bakiyesi ="+hesapYonetimi.getHesap());  
 hesapYonetimi.yatirma(100);  
 System.*out*.println("Hesap Bakiyesi ="+hesapYonetimi.getHesap());  
 try {  
 hesapYonetimi.cekme(90);  
 }catch (Exception exception){  
 System.*out*.println(exception.getMessage());  
 }  
 System.*out*.println("Hesap Bakiyesi ="+hesapYonetimi.getHesap());  
 try {  
 hesapYonetimi.cekme(20);  
 }catch (Exception exception) {  
 System.*out*.println(exception.getMessage());  
 }  
 System.*out*.println("Hesap Bakiyesi ="+hesapYonetimi.getHesap());  
  
 }

**Custom bır exceptıon oluşturmak ıcın bır tane class acıyorux ve exceptıonu extends edıyoruz.**

public class BakiyeYetersizException extends Exception{  
 String message;  
 public BakiyeYetersizException(String message){  
 this.message = message;  
 }  
  
 @Override  
 public String getMessage() {  
 return this.message;  
 }  
}

**Usttekı örnekte Hesap yonetımı kısmında sadece sunlar degısıyor**

public void cekme (double tutar) throws **BakiyeYetersizException** {  
 if (hesap>=tutar){  
 hesap = getHesap() - tutar;  
 }else {  
 throw new **BakiyeYetersizException**("Bakiye Yetersiz");  
 }

**Maın kısmında ıse sunlar**

try {  
 hesapYonetimi.cekme(90);  
}catch (**BakiyeYetersizException** exception){  
 System.*out*.println(exception.getMessage());  
}

**Yani hata classımızın adını verıyoruz.**

**Dosyalarla çalışmak**

Javanın dosya ile çalışmak için hazır classları var. İlk olarak file classını inceliycez.

File file = new File(“c:\\Javademos\\files\\students.txt”);

File.createNewfile(); -> Dosya oluşturur ama try catch içinde yazmak gerekiyor.

Örnek->

File file = new File("C:\\javademos\\files\\students.txt");  
try {  
 if(file.createNewFile())  
 {  
 System.*out*.println("Dosya oluşturuldu");  
 }else{  
 System.*out*.println("Dosya zaten var");  
 }  
}  
catch (IOException e) {  
 e.printStackTrace(); //kırmızı hata verır.

File.exist varsa true yoksa false dönüyor.

}  
public static void getFileInfo(){  
 File file = new File("C:\\javademos\\files\\students.txt");  
 if(file.exists()){  
 System.*out*.println("Dosya adı: "+file.getName());  
 System.*out*.println("Dosya yolu: "+file.getPath());  
 System.*out*.println("Dosya okunabilir mi: "+file.canRead());  
 System.*out*.println("Dosya okunabilir mi: "+file.canWrite());  
 System.*out*.println("Dosya okunabilir mi: "+file.length());  
 }  
}

Dosyayı okumak için

public static void readFile(){  
 File file = new File("C:\\javademos\\files\\students.txt");  
 try {  
 Scanner reader = new Scanner(file);  
 while (reader.hasNextLine()){  
 String line = reader.nextLine();  
 System.*out*.println(line);  
 }  
 reader.close();  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
  
}

Dosyaya yazmak

public static void writeFile(){  
 try {  
 BufferedWriter writer = new BufferedWriter(new FileWriter("C:\\javademos\\files\\students.txt",true));  
 writer.newLine();  
 writer.write("Ahmet");  
 System.*out*.println("Dosyaya Yazıldı");  
 writer.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }

Generics

Repository üzerinden çalışıcaz. Genelde veritabanı işleminde kullanılır bu desen

IEntity Repository adında bir interface açıyoruz.

public interface IEntityRepository<T extends IEntity> {  
 void add(T entity);  
 void delete(T entity);  
 void update(T entity);  
}

Sonrasında CustomerDal classımızıa bunu implement ediyoruz.

public class CustomerDal implements IEntityRepository<Customer>{  
  
 @Override  
 public void add(Customer entity) {  
  
 }  
  
 @Override  
 public void delete(Customer entity) {  
  
 }  
  
 @Override  
 public void update(Customer entity) {  
  
 }  
}

<Customer> olarak girdiğimiz için ovverride edince Customerla ilgili veri tabanı kodlarını yazacağımız açıkça görülmektedir. Aynısını product ıcınde yapıyoruz onada <Product> yazıyoruz.

Birde Generic method var o da bir classı direkt olarak generik yapmaz içerisindeki methodu yapar.

Public class validator{

Public <T> void validate(T entity){

}

Mainde çağırıp içine customer ekleyebiliriz.

Public static void main(String[]args){

Validator validator = new Validator();

Customer customer = new Customer();

Validator.validate(Customer);

}

Threading

Tek threadde kullanıcı bir işlemin bitip diğerinin başlamasını bekler. Multi thread aynı anda birden fazla işlemin gerçekleşmesini sağlar.

Bu örnekte kronometre ustunden threading yapıcaz maınde 3 kere cagırıcaz hepsı aynı anda calısıcak.

public class KronometreThread implements Runnable{  
  
 *//standart kod olusturmak ıcın* private Thread thread;  
 private String threadName;  
  
 public KronometreThread(String threadName){  
 this.threadName = threadName;  
 System.*out*.println("Oluşturuluyor: "+threadName);  
 }  
 @Override  
 public void run() {  
 System.*out*.println("Çalıştırılıyor : "+threadName);  
 *//sadece burayı kullanmak ıstedıgımız sekılde yazıcaz* try{  
 for (int i=1;i<=10;i++){  
 System.*out*.println(threadName + " : "+i);  
 Thread.*sleep*(1000);  
 }  
 }catch (InterruptedException exception){  
 System.*out*.println("Kesildi : "+threadName);  
 }  
  
 System.*out*.println("Thread bitti : "+ threadName);  
  
 }  
  
 *//standart kod başlatmak ıcın* public void start(){  
 System.*out*.println("Thread nesnesi oluşuyor");  
 if(thread==null){  
 thread = new Thread(this,threadName);  
 thread.start();  
 }  
 }  
  
}

Mainde 3 kere çağırıyoruz.

public class Main {  
  
 public static void main(String[] args) {  
 KronometreThread thread1 = new KronometreThread("thread1");  
 KronometreThread thread2 = new KronometreThread("thread2");  
 KronometreThread thread3 = new KronometreThread("thread3");  
  
 thread1.start();  
 thread2.start();  
 thread3.start();  
 }  
}

JDBC İle veri tabanı

Mysql ile çalışıcaz

Temel sql sorguları

Select \* from country -> Bize country tablosunu getiriyor.

Select \* from country where continent =’Asia’

Country tablosundaki continenti asyaya eşit olanları getirir.

\*kullandığımızda hepsini getirir. Fakat filtreleme yapabiliriz.

Select Code,Name,Continent,Region from country

Bu sorguda sadece yazdığımız kolonlar gelir

İnsert ile tabloya bir şey ekleyebiliyoruz.

İnsert into city (Name,Countrycode,District,population) values (‘Düzce’,’Tur’,’Duzce’,50000)

Hangi tablo Hangi stunlar Eklenecekler

Update ile güncelleme yapıyoruz.

Update city set population=60000 where name =‘Düzce’

Hangi tablo Neyi güncelliycez İsmi düzce olanlar

Delete silmek için kullanıyoruz.

Korumaya alırsa SET SQL\_SAFE\_UPDATES = 0; yazıyoruz

Delete from city where id=4080

City tablosundaki idsi 4080 olan satırı komple siler

İlk bağlantıların oluşturulması

Hangi ideyi kullanıyorsak add jar diyerek googleden bakıyoruz.

Static String UserName=”root”;

Static String password=”12345”;

Static String dbUrl=”JDBC:mysql://localhost:3306”//…..”;

Public static void main(Strin[] args) throws SQLException{

Connection connection = null;

}

Try{

Connection = DriverManager.getConnetcion(dbUrl,userName,password);

System.out.println(“Bağlantı kuruldu”);

}catch(SQLException exception){

System.out.println(exception.getMessage());

}finally {connection.close();}

Eğer saat hatası verirse Mysql jdbc drive time zone ıssue bak stackoverflow

Her bağladığımız yerde bunu tekrar yazmamak için birtane dbHelper classı açıyoruz

public class DbHelper {  
 static String *dbUrl* = "jdbc:mysql://localhost:3306/world?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC";  
 static String *userName*= "root";  
 static String *password* = "12345";  
   
 public Connection getConnection() throws SQLException{  
 return DriverManager.*getConnection*(*dbUrl*,*userName*,*password*);  
 }  
   
 public void showErrorMessage(SQLException exception){  
 System.*out*.println(exception.getErrorCode());  
 System.*out*.println(exception.getMessage());  
 }  
}

Mainde bu kodu çağırıyoruz.

public static void main(String[] args) throws SQLException {  
 Connection connection = null;  
 DbHelper dbHelper = new DbHelper();  
  
 try{  
 connection = dbHelper.getConnection();  
 System.*out*.println("Bağlantı kuruldu");  
 }catch (SQLException exception){  
 dbHelper.showErrorMessage(exception);  
 }finally {  
 connection.close();  
 }  
  
  
  
}

SQL ile tabloyu listelerken

Connection connection = null;  
DbHelper dbHelper = new DbHelper();  
Statement statement = null; *//sql sorgu cümleciği*ResultSet resultSet; *//Gelen sonucun oluşturulması ıcın*try{  
 connection = dbHelper.getConnection();  
 statement = connection.createStatement();*//Bu bağlantı için oluştur.* resultSet=statement.executeQuery("select code,name from country");  
 while (resultSet.next()){  
 System.*out*.println(resultSet.getString("name"));  
 }  
 System.*out*.println("Bağlantı kuruldu");  
}catch (SQLException exception){  
 dbHelper.showErrorMessage(exception);  
}finally {  
 connection.close();  
}

Select ile verileri bir ArrayListe kaydetmek için->

Önce country adında bir calss oluşturuyouz

public class Country {  
 private String code;  
 private String name;  
 private String continent;  
 private String region;  
  
 public Country(String code, String name, String continent, String region){  
 this.code = code;  
 this.name = name;  
 this.continent = continent;  
 this.region = region;  
 }  
}

Mainde Select yapıyoruz.

public static void selectDemo() throws SQLException{  
 Connection connection = null;  
 DbHelper dbHelper = new DbHelper();  
 Statement statement = null;  
 ResultSet resultSet;  
 try{  
 connection = dbHelper.getConnection();  
 statement = connection.createStatement();  
 resultSet = statement.executeQuery("select Code,Name,Continent,Region from country ");  
 ArrayList<Country> countries = new ArrayList<Country>();  
 while (resultSet.next()){  
 countries.add(new Country(  
 resultSet.getString("code"),  
 resultSet.getString("name"),  
 resultSet.getString("continent"),  
 resultSet.getString("region")));  
 }  
 System.*out*.println(countries.size());  
  
 }catch (SQLException exception){  
 dbHelper.showErrorMessage(exception);  
 }finally {  
 connection.close();  
 }

}

İnsert ile kayıt ekleme işlemi yapma veriyi dışardan aldığımızı düşünüyoruz.

Selectle benzer kısımları var ama yınede hepsını bastan yazıyorum Statement yerıne prapered statement geliyor.

Connection connection = null;  
DbHelper dbHelper = new DbHelper();  
PreparedStatement statement = null;  
ResultSet resultSet;  
try {  
 connection = dbHelper.getConnection();  
 String sql = "insert into city(name,countrycode,district,population) values (?,?,?,?)";  
 statement = connection.prepareStatement(sql);  
 statement.setString(1, "Düzce3");  
 statement.setString(2, "Tur");  
 statement.setString(3, "Turkey");  
 statement.setString(4, "1");  
 statement.executeUpdate();  
 System.*out*.println("Kayıt eklendi");  
} catch (SQLException exception) {  
 dbHelper.showErrorMessage(exception);  
} finally {  
 statement.close();  
 connection.close();  
}

Update ile güncelleme işlemi Yine dışarıdan bir veri almış gibi davranıyoruz.

Connection connection = null;  
DbHelper dbHelper = new DbHelper();  
PreparedStatement statement = null;  
ResultSet resultSet;  
try {  
 connection = dbHelper.getConnection();  
 String sql="update city set population = 1000 where id= ?";  
 statement = connection.prepareStatement(sql);  
 statement.setInt(1,4082);  
 statement.executeUpdate();  
 System.*out*.println("Kayıt Güncellendi");  
} catch (SQLException exception) {  
 dbHelper.showErrorMessage(exception);  
} finally {  
 statement.close();  
 connection.close();  
}

Delete işlemi ile silme

Connection connection = null;  
DbHelper dbHelper = new DbHelper();  
PreparedStatement statement = null;  
ResultSet resultSet;  
try {  
 connection = dbHelper.getConnection();  
 String sql="delete from city where id= ?";  
 statement = connection.prepareStatement(sql);  
 statement.setInt(1,4081);  
 statement.executeUpdate();  
 System.*out*.println("Kayıt silindi");  
} catch (SQLException exception) {  
 dbHelper.showErrorMessage(exception);  
} finally {  
 statement.close();  
 connection.close();  
}