**Liskov Substitution Prensibi**

Liskov Substitution (Yerine Geçme) prensibi alt sınıfların üst sınıfın özelliklerini ve fonksiyonlarını aynı işlevi gösterecek şekilde kullanabilmesidir. Üst sınıf referansına ihtiyaç duyan bir fonksiyona gönderilen alt sınıfın fonksiyonun gereklerini yerine getirebilmesi gerekmektedir. Ayrıca üst sınıfta tanımlı olan özellik ve fonksiyonlar alt sınıf için gereksiz olmamalıdır.

Yerine geçme prensibinin sağlanmaması durumunda genellikle üst sınıfta ortak olmayan özellik ve fonksiyonlar yeni bir sınıfa aktarılır. Bu şekilde alt sınıf ihtiyacı olan sınıflardan özellik ve fonksiyonları miras alabilmektedir.

**Uygulama Kodları**

public interface Sekil {  
 double alanHesapla();  
 double cevreHesapla();  
}

public interface CevreHesap {  
 double cevreHesapla();  
}

public interface AlanHesap {  
 double alanHesapla();  
}

public class Kare implements CevreHesap, AlanHesap {  
  
 private double kenar;  
 public Kare(double kenar) {  
 this.kenar = kenar;  
 }  
 public Kare() { }  
  
 public double getKenar() {  
 return kenar;  
 }  
  
 public void setKenar(double kenar) {  
 this.kenar = kenar;  
 }  
  
 @Override  
 public double cevreHesapla() {  
 return 4 \* kenar;  
 }  
  
 @Override  
 public double alanHesapla() {  
 return kenar \* kenar;  
 }  
}

public class Dikdortgen implements CevreHesap, AlanHesap {  
  
 private double kenar1, kenar2;  
 public Dikdortgen(double kenar1, double kenar2) {  
 this.kenar1 = kenar1;  
 this.kenar2 = kenar2;  
 }  
 public Dikdortgen() { }  
  
 public double getKenar1() {  
 return kenar1;  
 }  
  
 public void setKenar1(double kenar1) {  
 this.kenar1 = kenar1;  
 }  
  
 public double getKenar2() {  
 return kenar2;  
 }  
  
 public void setKenar2(double kenar2) {  
 this.kenar2 = kenar2;  
 }  
  
 @Override  
 public double cevreHesapla() {  
 return 2 \* (kenar1 + kenar2);  
 }  
  
 @Override  
 public double alanHesapla() {  
 return kenar1 \* kenar2;  
 }  
}

public class Cember implements CevreHesap {  
  
 private double yaricap;  
 public Cember(double yaricap) {  
 this.yaricap = yaricap;  
 }  
 public Cember() { }  
  
 public double getYaricap() {  
 return yaricap;  
 }  
  
 public void setYaricap(double yaricap) {  
 this.yaricap = yaricap;  
 }  
  
 @Override  
 public double cevreHesapla() {  
 return 6 \* getYaricap();  
 }  
  
}

public class Toplayici {  
  
 public double alanToplayici(ArrayList<AlanHesap> sekiller) {  
 double toplam = 0;  
 for(AlanHesap sekil : sekiller) {  
 toplam += sekil.alanHesapla();  
 }  
 return toplam;  
 }  
  
 public double cevreToplayici(ArrayList<CevreHesap> sekiller) {  
 double toplam = 0;  
 for(CevreHesap sekil : sekiller) {  
 toplam += sekil.cevreHesapla();  
 }  
 return toplam;  
 }  
}

public class Main {  
  
 public static void main(String[] args) {  
  
 Toplayici toplayici = new Toplayici();  
 Kare kare1 = new Kare();  
 kare1.setKenar(4);  
 Kare kare2 = new Kare(5);  
 Dikdortgen dikdortgen = new Dikdortgen(4, 6);  
 Cember cember = new Cember(7);  
  
 ArrayList<CevreHesap> cevreHesapArrayList = new ArrayList<>();  
 ArrayList<AlanHesap> alanHesapArrayList = new ArrayList<>();  
  
 cevreHesapArrayList.add(kare1);  
 cevreHesapArrayList.add(dikdortgen);  
 cevreHesapArrayList.add(cember);  
  
 alanHesapArrayList.add(kare2);  
 alanHesapArrayList.add(dikdortgen);  
  
 System.*out*.println(toplayici.alanToplayici(alanHesapArrayList));  
 System.*out*.println(toplayici.cevreToplayici(cevreHesapArrayList));  
  
 }  
}