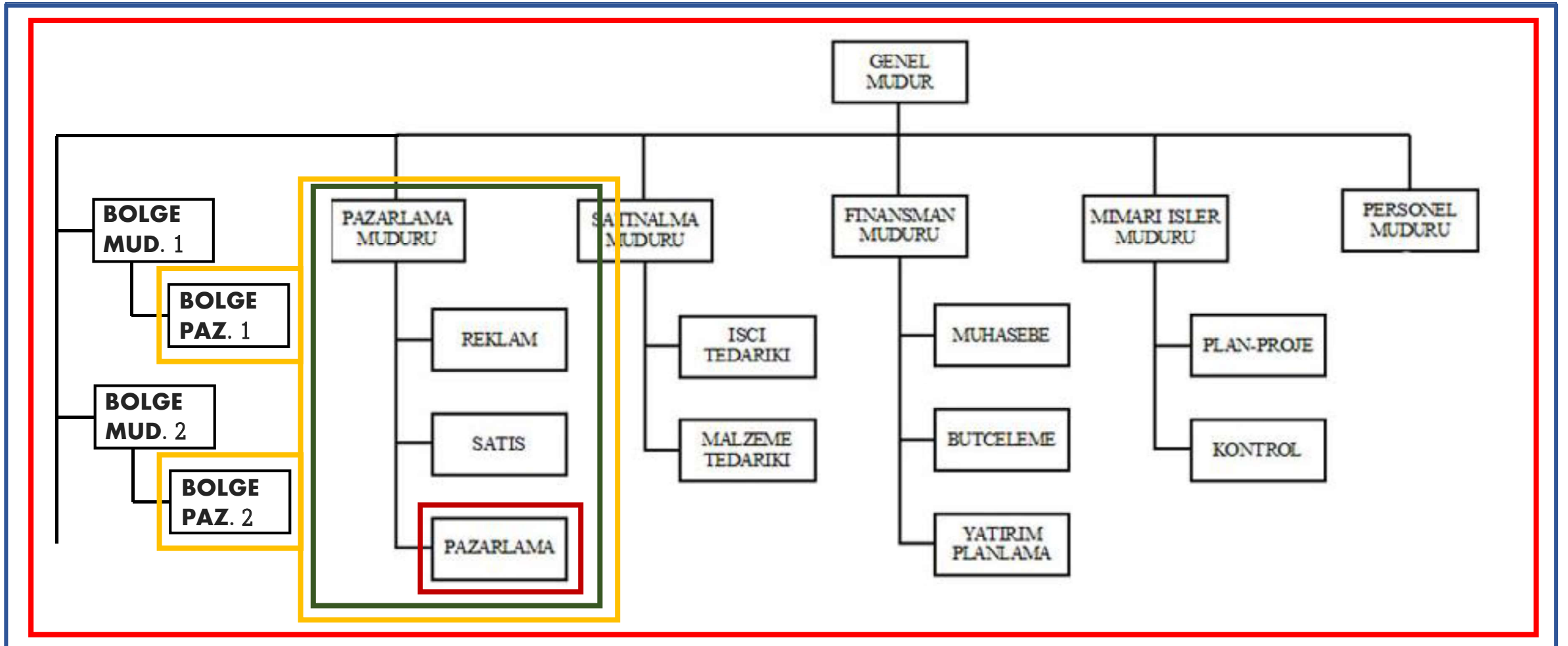




## Access Modifier

Java bir method'u, variable'i ya da class'i oluşturulurken bu öğelere kimlerin erişebileceğini belirtme olanağı tanır.





## Access Modifier

Java bir method'u, variable'i ya da class'i oluşturulurken bu öğelere kimlerin erişebileceğini belirtme olanağı tanır.

Bu işlemi gerçekleştirebilmek için kullanılan anahtar kelimelere **Access Modifiers** (Erişim Belirleyiciler) adını veririz.

Java'da 4 Farklı access modifier vardır

- 1) Private
- 2) Default (Eğer herhangi bir Access Modifier yazmazsak, Java Access Modifier'i default olarak kabul eder.)
- 3) Protected
- 4) Public

**NOT :** Class'lar için sadece public veya default kullanılabilir. Class'lar private veya protected olamazlar.



# Access Modifier

1) Private : Sadece olusturuldugu Class'da kullanilabilir

```
1 package package1;
2
3 public class Class1 {
4
5     private static String privateVariable;
6     private static void privateMethod(){
7     }
8
9     public static void main(String[] args) {
10
11
12     }
13
14     public static void method1(){
15
16
17     }
18
19     public void method2(){
20
21
22     }
```



## Access Modifier

2) Default : Sadece olusturuldugu Class'in ait oldugu Package icinde kullanilabilir

The screenshot illustrates the default access modifier in Java. On the left, the 'src' directory contains four packages: package1, package2, package3, and package4. package1 is highlighted with a green box and a green checkmark, indicating it is the package where the class is defined. package2, package3, and package4 are grouped by a red box with a large red 'X' over it, indicating they are not accessible to the class. A blue arrow points from package1 in the package explorer to the code editor. The code editor shows the following code for Class1:

```
1 package package1;
2
3 public class Class1 {
4
5     static String defaultVariable;
6     static void defaultMethod(){
7     }
8
9     public static void main(String[] args) {
10
11     }
12
13     public static void method1(){
14
15     }
16
17     public void method2(){
18
19     }
20
21
22 }
23 }
```

The code is enclosed in a green box with a large green checkmark, indicating it is correct. The package name 'package1' is highlighted in blue, and the class name 'Class1' is highlighted in blue. The package explorer on the left shows the package structure, with package1 being the package where the class is defined.



## Access Modifier

- 3) Protected : Olusturulduđu Class'ın ait olduđu Package icinde ve baska package icindeki Child Class'larda kullanılabilir

```
1 package package1;
2
3 public class Class1 {
4
5     protected static String protectedVariable;
6     protected static void protectedMethod(){
7     }
8
9     public static void main(String[] args) {
10
11     }
12
13     public static void method1(){
14
15     }
16
17     public void method2(){
18
19     }
20
21
22 }
23
```



## Access Modifier

4) Public : Her yerden erişilebilir. Hicbir sinirlama (restriction) icermez.

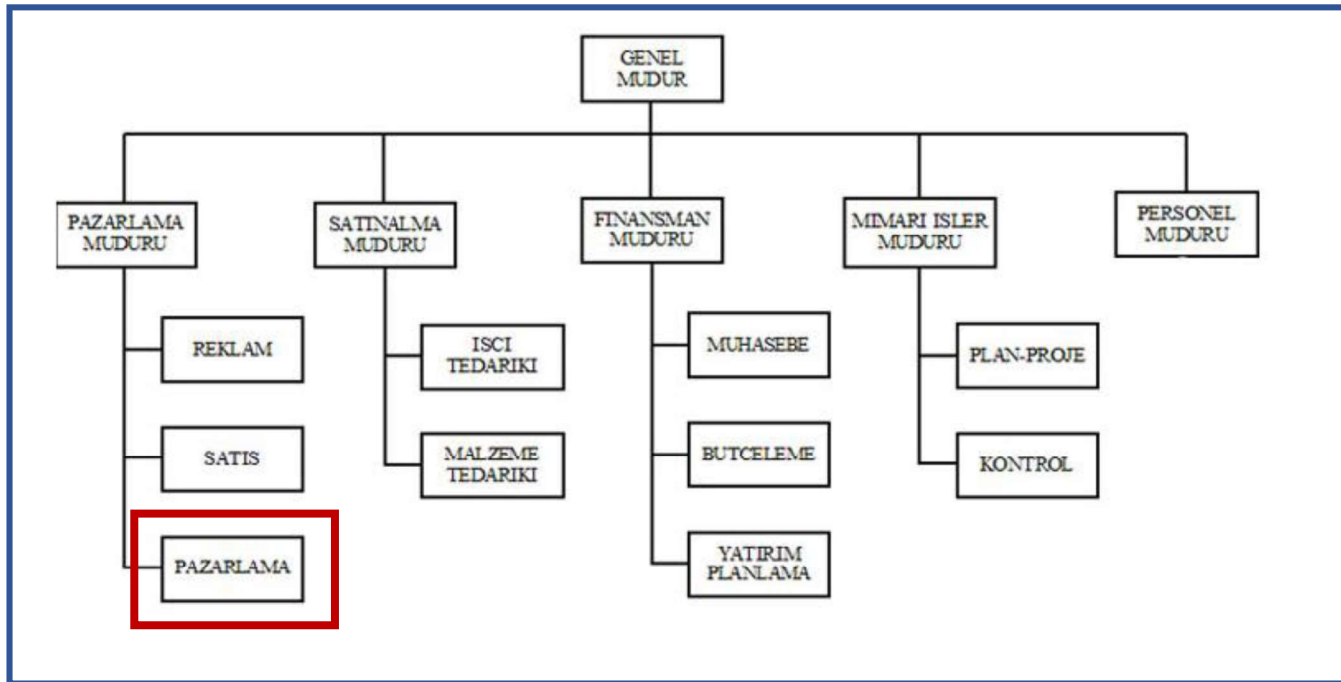
The screenshot displays an IDE interface with a project structure on the left and a code editor on the right. The project structure shows a hierarchy of packages (package1, package2, package3, package4) and classes (Class1, Class2, ClassChild1, ClassChild2). A green checkmark is placed over the package3 structure. The code editor shows the following code:

```
1 package package1;
2
3 public class Class1 {
4
5     public static String publicVariable;
6     public static void publicMethod(){
7     }
8
9     public static void main(String[] args) {
10
11
12     }
13
14     public static void method1(){
15
16
17     }
18
19     public void method2(){
20
21
22     }
23 }
```

A blue box highlights the class-level code (lines 3-7), and a green box highlights the method-level code (lines 9-22). A large green checkmark is placed over the method-level code.



# Encapsulation (Data Hiding)



Pazarlama birimi olarak rapor duzenliyoruz  
Ihtiyaclarimiz

- 1- Satis bolumundeki personel rapor hazirlayacagimiz verileri girsin ama rapor sonuclarini goremesin
- 2- Raporu gormesine izin verilenler raporu gorsun ama degistiremesin

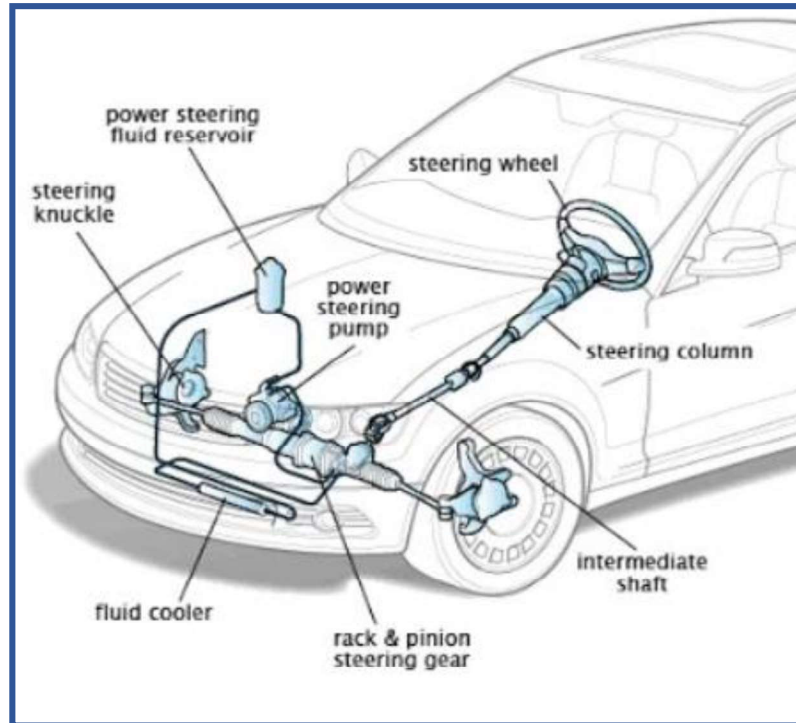
Java bir method'u ya da variable'i olusturulurken class disindan bu oğelere erişimi kisitlama veya belirlenmis datolari degistirebilme olanağı tanir.





# Encapsulation (Data Hiding)

Before Encapsulation



After Encapsulation (Data Gizleme)







## Encapsulation (Data Hiding)

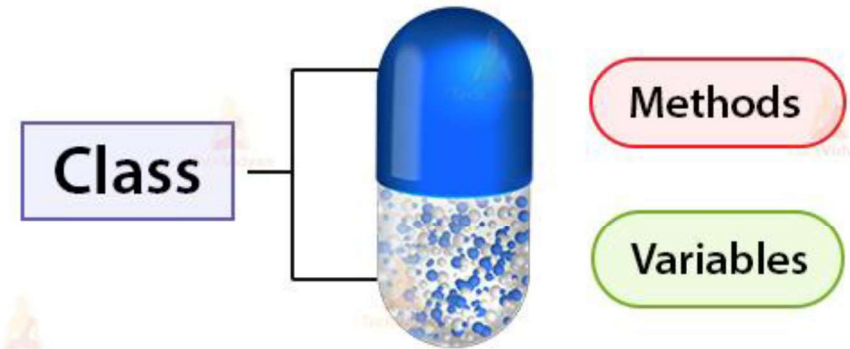
**Encapsulation**, önemli Class üyelerini korumak için uygulanan data saklama yöntemidir.

Farklı Class'lardan erişilerek ya da yanlış kullanım sonucu kodunuzun veya önemli datalarınızın değişmesini istemiyorsanız Encapsulation ile datalarınızı koruyabilirsiniz.

Encapsule edilen variable ve method'lara sadece sizin verdiğiniz oranda erişilebilir.

Encapsule edilen variable ve method'lara izin verdiğiniz kişiler ulaşabilir ama DEĞİSTİREMEZ.

### Encapsulation in Java





## Encapsulation (Data Hiding)

Datalarimizi korumak için data'larimizi private yapabiliriz ama private yaptigimiz datalari baska Class'lar kullanamaz. Bu durum OOP concept'ine uygun olmaz.

Private yaparak KORUMA ALTINA aldigimiz Class uyeleri'ninin sadece OKUNMASINI istiyorsak `getter()`, DEGER ATANMASINA da izin vermek istiyorsak `setter()` method'larini olustururuz.

Encapsulation iki adimda yapilir:

- 1) Class uyelerini (*variable, method*) private yapmalisiniz.
- 2) public olan `getter()` ve `setter()` methodlar uretmelisiniz.

**Not:** `getter()` data'yi sadece okumamiza yarar, data'da degisiklik yapamaz.

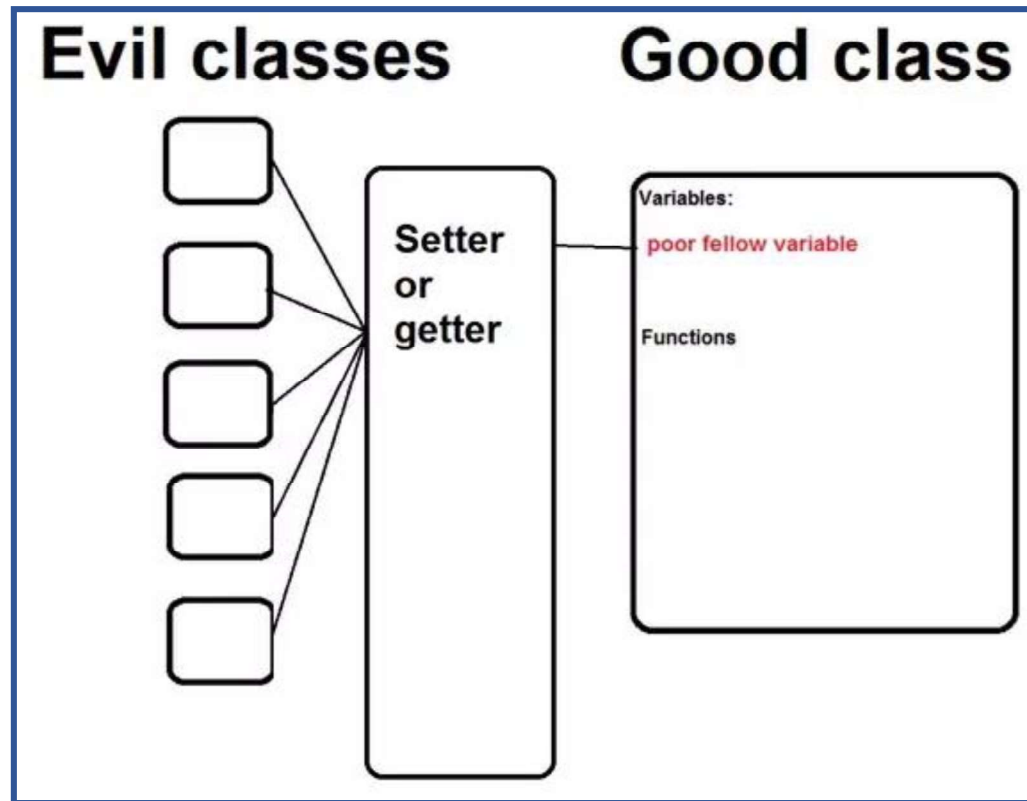
**Not:** `setter()` baska Class'larda olusturulan objeler icin data degerini degistirmemizi saglar.



# Getters & Setters



Getters and setters  
lead to the dark side...

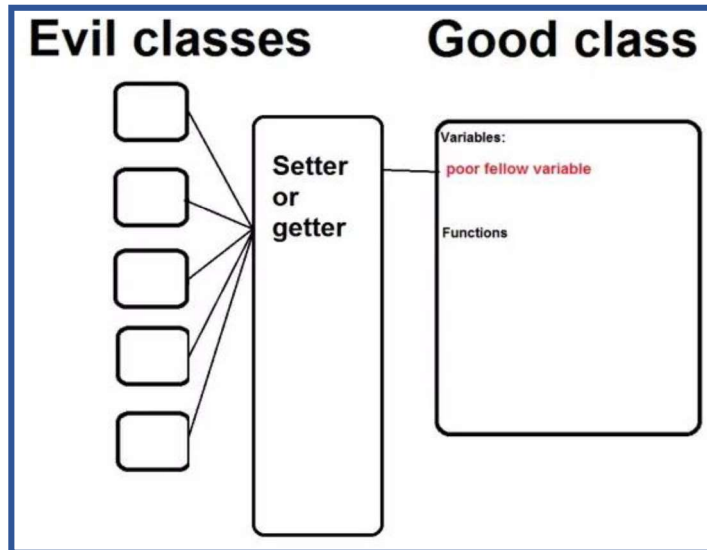




# Getters & Setters



Getters and setters  
lead to the dark side...



**Setter Methods :** Encapsule ettigimiz class uyelerinin,baska class'lardan obje uretilerek deger atanmasina izin verir.

```
public class Example {  
  
    private String tcNo= "12345678901";  
  
    public static void main(String[] args) {  
  
    }  
  
    public void setTcNo(String tcNo) {  
        this.tcNo = tcNo;  
    }  
}
```

Eger sadece **setter** method olusturulursa data degerleri degistirilebilir ama ilk atanan deger veya bizim atadigimiz yeni deger baska class'lardan okunamaz.



# Getters & Setters (Java Beans)

Getter and setter method'lari "**Java Beans**" olarak da adlandırılır.

**Isim verme kurallari** (Naming convention)

1) Data type'lari **boolean** olan variable'lerin getter metod isimleri "**is**" ile baslar.

```
private boolean happy = true;

public boolean isHappy() {
    return happy;
}
```

```
private int a=10;

public int getA() {
    return a;
}
```



# Getters & Setters (Java Beans)

Isim verme kurallari (Naming convention)

```
private int a=10;  
private boolean happy;  
  
public void setA(int a) {  
    this.a = a;  
}  
  
public void setHappy(boolean happy) {  
    this.happy = happy;  
}
```





## Tekrar Sorulari

1) **Encapsulation** nedir?

Encapsulation, hassas datalari korumak icin kullanılan data saklama yontemidir

2) Datalari nasil saklariz?

Data'lari private access modifier kullanarak saklariz.

3) Saklanan datalara diger class'lardan ulasabiliriz?

Getter ve setter method'larini kullanarak ulasabiliriz.

4) **getter( )** method'u ne yapar ?

Saklanan datalari okumamizi saglar.

5) **setter ( )** method'u ne yapar ?

Saklanan datalari obje uzerinden update edebilmemizi saglar

6) **immutable class** nedir?

Encapsule edilen bir class'da sadece getter method'u olusturursak datalari okuyabiliriz ama degistiremeyiz. Bu tur class'lara immutable class denir.

7) **setter( )** method'lari icin **naming convention** nedir?

Tum data turleri icin isimler "set" ile baslar.

8) **getter( )** method'lari icin **naming convention** nedir?

Boolean data turu icin "is" ile, diger data turleri icin "get" ile baslar.



## Getters & Setters

Which are methods using JavaBeans naming conventions for accessors and mutators?  
(Choose all that apply)

- A. `public boolean getCanSwim() { return canSwim;}`
- B. `public boolean canSwim() { return numberWings;}`
- C. `public int getNumWings() { return numberWings;}`
- D. `public int numWings() { return numberWings;}`
- E. `public void setCanSwim(boolean b) { canSwim = b;}`



## Getters & Setters

Which of the following are true? (Choose all that apply)

- A.** Encapsulation uses package private instance variables.
- B.** Encapsulation uses private instance variables.
- C.** Encapsulation allows setters.
- D.** Immutability uses package private instance variables.
- E.** Immutability uses private instance variables.
- F.** Immutability allows setters.