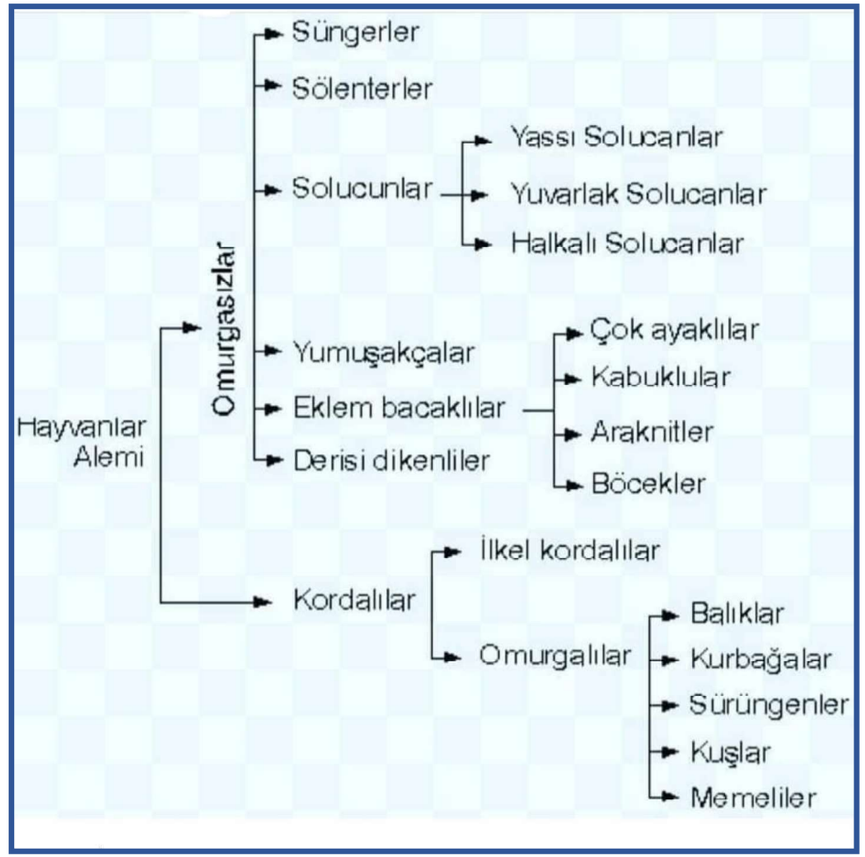
Abstract Classes

Abstract class, genellikle ortak özellikleri olan nesneleri tek bir çatı altında toplamak için kullanılır.

Tum child class'larda olmasini istedigimiz dinamik ozellikleri (methods) abstract class'da abstract method olarak olustururuz.

Abstract olmayan (concrete) tum child class'lar abstract method'lari override etmek zorundadir. Boylece tum child class'lar ayni dinamik ozelliklere (methods) sahip olurlar.



Abstract Classes

Abstract class nasil olusturulur ?

Abstract class olusturmak icin class keyword'unden once abstract keyword'u yazilmalidir.

```
public abstract class Personel {

}
```

Abstract method nasil olusturulur ?

Abstract method olusturmak icin abstract keyword'u yazilmalidir.

Abstract method'un body'si olmaz (No implementation), body olusturursaniz CTE olusur.

Abstract method private, final veya static olarak tanimlanamaz.

```
public abstract void maasHesapla();

public abstract void mesaiBilgisi() {

}
```

Abstract Classes

NOT : Bir abstract class, abstract veya concrete method'lara sahip olabilir

```
public abstract class Personel {

    public String isim;
    public int maas;

    public abstract void maasHesapla();
    public abstract void mesaiBilgisi();

    public void ozelSigorta() {
        System.out.println("Bu personel ozel sigorta kapsamindadir");
    }
}
```

Concrete class icinde Abstract method OLUSTURULAMAZ

```
public class IdariPersonel extends Personel{

    public static void main(String[] args) {


    }

    public abstract void vardiya();
```

Abstract Classes

```java
public abstract class Personel {

    public String isim;
    public int maas;

    public abstract void maasHesapla();
    public abstract void mesaiBilgisi();

    public void ozelSigorta() {
        System.out.println("Bu personel ozel sigorta kapsamindadir");
    }
}
```

```java
public class IdariPersonel extends Personel{

    public static void main(String[] args) {


    }
}
```

```java
public class Isci extends Personel{

    public static void main(String[] args) {


    }

    @Override
    public void maasHesapla() {
        System.out.println("maas : "+ 30*8*15);
    }
    @Override
    public void mesaiBilgisi() {
        System.out.println("Isciler 8 saat mesai yapar");
    }
}
```

```java
public abstract class YanHizmetler extends Personel{

    public static void main(String[] args) {

        YanHizmetler obj1 = new YanHizmetler();
    }

}
```

Abstract Classes

Kural 1)

Concrete bir child class, parent'i olan abstract class'lardaki tum abstract method'lari implement etmelidir, diger turlu CTE olusur.

Kural 2)

Abstract method'lari implement etmek icin oncelikle overriding yapilmalidir.

Kural 3)

Abstract class, abstract olmayan (concrete) method'lara da sahip olabilir. Cocrete method'lar implement edilmek ZORUNDA DEGILDIR. Parent-child iliskisi ile kullanilabilirler veya istenirse override edilebilirler

```java
public abstract class Personel {

    public String isim;
    public int maas;

    public abstract void maasHesapla();
    public abstract void mesaiBilgisi();

    public void ozelSigorta() {
        System.out.println("Bu personel ozel sigorta kapsamindadir");
    }
}
```

```java
public class Isci extends Personel{

    public static void main(String[] args) {
        Isci isci1=new Isci();
        isci1.isim="Mehmet";
        System.out.println(isci1.isim);
        isci1.ozelSigorta();
        isci1.maasHesapla();
    }

    @Override
    public void maasHesapla() {
        System.out.println("maas : "+ 30*8*15);
    }
    @Override
    public void mesaiBilgisi() {
        System.out.println("Isciler 8 saat mesai yapar");
    }
```

Abstract Classes

Kural 4)

Abstract bir child class, parent'i olan abstract class'lardaki method'lari implement etmek ZORUNDA DEGILDIR. Parent-child iliskisi ile tum method'lara ulasabilir.

```
public abstract class Personel {

    public String isim;
    public int maas;

    public abstract void maasHesapla();
    public abstract void mesaiBilgisi();

    public void ozelSigorta() {
        System.out.println("Bu personel ozel sigorta kapsamindadir");
    }
}
```

Kural 5)

Abstract class'lar somutlastirilamaz (can not be instantiated) yani abstract class'larda obje olusturulamaz.

```
public abstract class YanHizmetler extends Personel{

    public static void main(String[] args) {

        YanHizmetler obj1 = new YanHizmetler();
    }

}
```

Kural 6)

Abstract class'lar private veya final olarak tanimlanamaz

Abstract Classes

Asagidaki kod calistirilirsa compile time error olur mu ?

```
public abstract class Animal {
  public abstract String getName();
}
```

```
public abstract class BigCat extends Animal {
  public abstract void roar();
}
```

```
public class Lion extends BigCat {
  public String getName() {
    return "Lion";
  }
  public void roar() {
    System.out.println("The Lion lets out a loud ROAR!");
  }
}
```

Abstract Classes

Asagidaki kod calistirilirsa compile time error olur mu ?

```
public abstract class Animal {
  public abstract String getName();
}
```

```
public abstract class BigCat extends Animal {
  public String getName() {

  return "BigCat";
}
public abstract void roar();
}
```

```
public class Lion extends BigCat {
  public void roar() {
    System.out.println("The Lion lets out a loud ROAR!");
  }
}
```

Interfaces

- Interface bir class degildir. Interface interface'dir

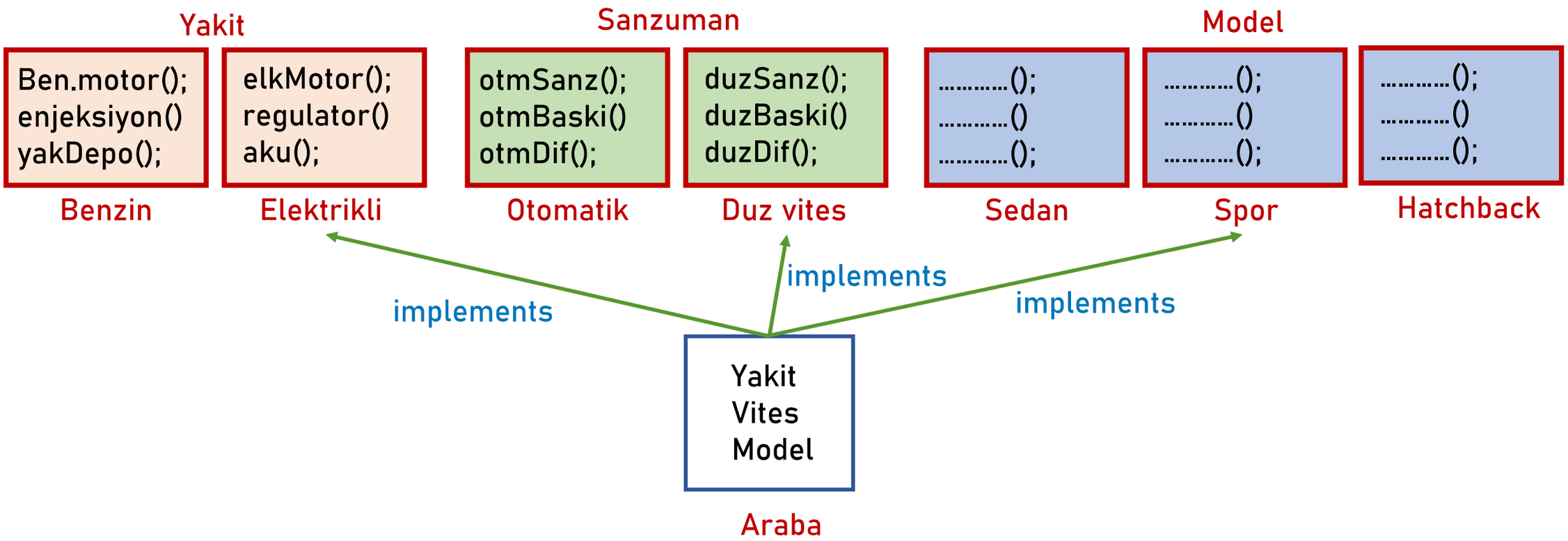
- Interface içinde sadece kendisinden türeyen sınıfların içini doldurmak zorunda olduğu, body'si olmayan method'larin olusturuldugu bir yapıdır.

- Kısacası kendisini inherit eden class'lar için, yerine getirmeleri gereken metodları belirten "tum alanlari doldurulmak zorunda olan bir sablon" gibidir.

- Interface'ler somutlastirilamaz (can not be instantiated) yani Interface'de obje olusturulamaz.

Interfaces

- Interface bir cesit to do list'dir. Concrete class'lari interface'deki tum metodlari implement etmek zorunda birakir. Nasil yapilacagina degil ne yapilacagina odaklanir
- Bir class birden fazla class'a extend edilemez ama birden fazla interface'e implement edilebilir.



Abstract Classes vs Interface

Abstract classes

1) Class'dir
2) abstract ve concrete method'lar konabilir
3) Kismi olarak abstraction saglar.
4) Birden fazla abstract class'a direk child olunamaz. Coklu inheritance'i desteklemez.
5) Hiz acisindan avantajlidir
6) Icindeki tüm nesnelerin "public" olması zorunlu degildir
7) soyut olmayan metodlar static olarak tanımlanabilir.
8) Abstract class constructor'a sahiptir

Interface

1) Class degildir.
2) sadece abstract method'lar konabilir.
3) Tam abstraction (soyutluk) saglar
4) Bir class'dan istediginiz kadar interface'i inherit edebilirsiniz. Coklu inheritance'a uygundur.
5) Hiz acisindan abstract class'a gore yavastir.
6) Icindeki tüm nesnelerin "public" olması gerekir
7) metodlar static olamaz
8) Interface constructor'a sahip degildir

Interfaces

```
public interface Interface01 {

    void vites();
    public void aku();
    abstract void teker();
    public abstract void motor();
}
```

- Interface'e sadece abstract public method'lar konabilir.

  Yandaki farkli yazimlar interface icin ayni seylerdir

- Return type'lar farkli olabilir

- Interface icindeki variable'lar mutlaka public, static , ve final olmalidir. private veya protected variable'lar compile time error verir.

  Yandaki farkli yazimlar interface icin ayni seylerdir

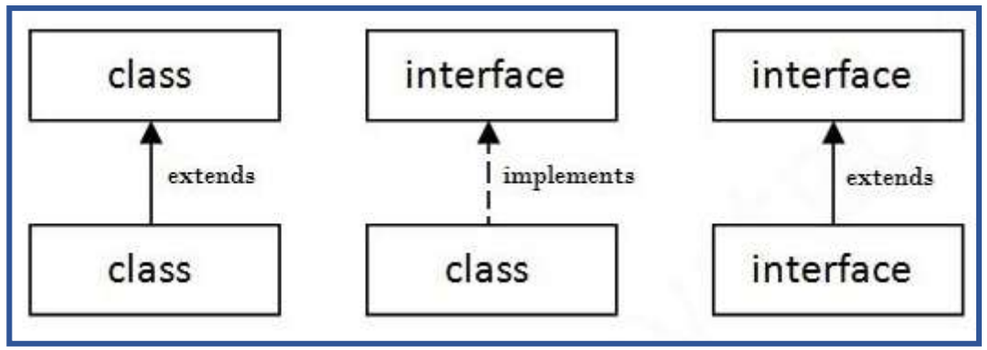
```
public interface Interface01 {

    int SAYI1=10;
    public int SAYI2=20;
    public static int SAYI3=30;
    public static final int SAYI4=40;
}
```

- Interface icindeki variable'lari mutlaka initialize etmek zorundasiniz, aksi takdirde Compile Time Error alirsiniz.

  int a = 12; gibi yapmalisiniz.

Interface'ler Icin Inheritance Kurallari

1) Interface'lerde inheritance yapmak icin implements keyword'u kullanilir.



2) Bir class birden fazla Interface'e implements ile baglanabilir

```
public class Arabam implements ElektrikMotor, OtomatikVites, Sedan{

}
```

Interface'ler Icin Inheritance Kurallari

3) Birden fazla Interface'i implements ile inherit ettigimizde ayni isimde variable veya method'larla karsilasabiliriz.

Bu durumda Java ne yapacagini net olarak bilmek isteyeceginden istedigimiz variable ismini interface ismi ile birlikte yazariz.

```
public class Arabam implements ElektrikMotor, OtomatikVites, Sedan {

    public static void main(String[] args) {
        System.out.println("Kasa Model no : " + Sedan.MODELNO);
        System.out.println("Vites Model no : " + OtomatikVites.MODELNO);
        System.out.println("Motor Model no : " + ElektrikMotor.MODELNO);
    }
}
```

Method'lar mecburen override yapilacagi icin hangi interface'den alindiginin hicbir onemi yoktur.

```
@Override
public void yakitTuru() {


}
```

Interface'ler Icin Inheritance Kurallari

4) Ayni isme fakat farkli return type'a sahip methodlari olan Interface'leri ayni class'dan inherit edemeyiz .

```
public interface ElektrikMotor {
    int MODELNO=10;

    void yakitTuru();
}
```

```
public interface OtomatikVites {
    int MODELNO=20;

    String yakitTuru();
}
```

implements

implements

```
3 public class Arabam implements ElektrikMotor, OtomatikVites, Sedan {
4
5     public static void main(String[] args) {
6
7     }
8
9 }
```

Interface'lerde Body'si Olan Method Yazilabilir mi ?

Java8'e kadar interface icinde "body'si olan method" olusturulamiyordu. Java8 ve uzeri versiyonlarda bu opsiyonu ekledi.

A) method'un basina "default" keyword'u kullanabilirsiniz.

```
public interface I05 {

    default int add(int a, int b) {
        return a+b;
    }
}
```

Burada kullandigimiz "default" access modifier degil, method turudur. (asagida gorulecegi uzere method'un access modifier'I public olarak belirtmisken, Java default kelimesini kullanmamiza izin veriyor)

```
public interface I05 {

    public default int add(int a, int b) {
        return a+b;
    }
}
```

Interface'lerde Body'si Olan Method Yazilabilir mi ?

B) return type'dan once "static" keyword'u kullanabilirsiniz.

```
public interface I05 {

    public static int add(int a, int b) {
        return a+b;
    }
}
```

NOT : default veya static keyword'u ile olusturulan method'lar override yapilmak zorunda degildir.

Interface'lerde Body'si Olan Method Yazilabilir mi ?

Soru : Interface'de kullanabildigimiz "static method" ve "default method" larin farki nedir?.

Cevap : Default keyword'u ile olusturulan method'lari obje kullanarak cagirabiliriz ama static olanlari inherit ettigimiz interface'den ismi ile cagirabiliriz.

```
public interface I05 {

    public static int add(int a, int b) {
        return a+b;
    }
}
```

```
public class Runner implements I05{

    public static void main(String[] args) {

        I05.add(10, 20);

    }
```

```
public interface I05 {

    public default int topla(int a, int b) {
        return a+b;
    }
}
```

```
public class Runner implements I05{

    public static void main(String[] args) {

        Runner rnr=new Runner();
        System.out.println(rnr.topla(20, 40));
    }
}
```

Abstract Class - Interface Tekrar Sorulari

1) Abstract class kullanilarak obje olusturamayiz. True/~~False~~

2) Interface kullanilarak obje olusturabiliriz. ~~True~~/False

3) public abstract int sumOfTwo(int n1, int n2) { } syntax olarak dogru mudur ? ~~True~~/False

4) public class Animal { public abstract void sound() } syntax olarak dogru mudur ? ~~True~~/False

5) public abstract class Animal {
public abstract void eat();
public void sound(){ } } syntax olarak dogru mudur ? True/~~False~~

6) Abstract class final olarak tanimlanamaz. True/~~False~~

7) public abstract class Animal {
private abstract void sound(); } syntax olarak dogru mudur ? ~~True~~/False

8) Java birden fazla class'a extends ile inherit etmenize izin vermez. True/~~False~~

9) Java birden fazla interface'e implements ile inherit etmenize izin verir. True/~~False~~

10) Bir interface abstract veya concrete method'lara sahip olabilir. ~~True~~/False

Interfaces

Soru 1)

Which of the following statements can be inserted in the blank line so that the code will compile successfully? (Choose all that apply)

```
public interface CanHop {}
public class Frog implements CanHop {
   public static void main(String[] args) {
     __________ frog = new TurtleFrog();
   }
}
public class BrazilianHornedFrog extends Frog {}
public class TurtleFrog extends Frog {}
```

A. `Frog`
B. `TurtleFrog`
C. `BrazilianHornedFrog`
D. `CanHop`
E. `Object`
F. `Long`

A, B, D, E. The blank can be filled with any class or interface that is a supertype of `TurtleFrog`. Option A is a superclass of `TurtleFrog`, and option B is the same class, so both are correct. `BrazilianHornedFrog` is not a superclass of `TurtleFrog`, so option C is incorrect. `TurtleFrog` inherits the `CanHope` interface, so option D is correct. All classes inherit `Object`, so option E is correct. Finally, `Long` is an unrelated class that is not a superclass of `TurtleFrog`, and is therefore incorrect.

Interfaces

Soru 2)

Choose the correct statement about the following code:

```
1: interface HasExoskeleton {
2:   abstract int getNumberOfSections();
3: }
4: abstract class Insect implements HasExoskeleton {
5:   abstract int getNumberOfLegs();
6: }
7: public class Beetle extends Insect {
8:   int getNumberOfLegs() { return 6; }
9: }
```

A. It compiles and runs without issue.

B. The code will not compile because of line 2.

C. The code will not compile because of line 4.

D. The code will not compile because of line 7.

E. It compiles but throws an exception at runtime.

D. The code fails to compile because `Beetle`, the first concrete subclass, doesn't implement `getNumberOfSections()`, which is inherited as an abstract method; therefore, option D is correct. Option B is incorrect because there is nothing wrong with this interface method definition. Option C is incorrect because an abstract class is not required to implement any abstract methods, including those inherited from an interface. Option E is incorrect because the code fails at compilation-time.

Interfaces

Soru 3)

Choose the correct statement about the following code:

```
1: public interface Herbivore {
2:    int amount = 10;
3:    public static void eatGrass();
4:    public int chew() {
5:      return 13;
6:    }
7: }
```

A. It compiles and runs without issue.

B. The code will not compile because of line 2.

C. The code will not compile because of line 3.

D. The code will not compile because of line 4.

E. The code will not compile because of lines 2 and 3.

F. The code will not compile because of lines 3 and 4.

F. The interface variable amount is correctly declared, with `public` and `static` being assumed and automatically inserted by the compiler, so option B is incorrect. The method declaration for `eatGrass()` on line 3 is incorrect because the method has been marked as `static` but no method body has been provided. The method declaration for `chew()` on line 4 is also incorrect, since an interface method that provides a body must be marked as `default` or `static` explicitly. Therefore, option F is the correct answer since this code contains two compile-time errors.

Interfaces

Soru 4)

Choose the correct statement about the following code:

```
1: public interface CanFly {
2:   void fly();
3: }
4: interface HasWings {
5:   public abstract Object getWindSpan();
6: }
7: abstract class Falcon implements CanFly, HasWings {
8: }
```

A. It compiles without issue.

B. The code will not compile because of line 2.

C. The code will not compile because of line 4.

D. The code will not compile because of line 5.

E. The code will not compile because of lines 2 and 5.

F. The code will not compile because the class `Falcon` doesn't implement the interface methods.

A. Although the definition of methods on lines 2 and 5 vary, both will be converted to `public abstract` by the compiler. Line 4 is fine, because an interface can have `public` or default access. Finally, the class `Falcon` doesn't need to implement the interface methods because it is marked as `abstract`. Therefore, the code will compile without issue.

Interfaces

Soru 5)

Which statements are true for both abstract classes and interfaces? (Choose all that apply)

A. All methods within them are assumed to be abstract.

B. Both can contain `public static final` variables.

C. Both can be extended using the `extend` keyword.

D. Both can contain default methods.

E. Both can contain static methods.

F. Neither can be instantiated directly.

B, C, E, F. Option A is wrong, because an abstract class may contain concrete methods. Since Java 8, interfaces may also contain concrete methods in form of static or default methods. Although all variables in interfaces are assumed to be `public static final`, abstract classes may contain them as well, so option B is correct. Both abstract classes and interfaces can be extended with the `extends` keyword, so option C is correct. Only interfaces can contain default methods, so option D is incorrect. Both abstract classes and interfaces can contain static methods, so option E is correct. Both structures require a concrete subclass to be instantiated, so option F is correct.

Interfaces

Soru 6)

Which statements are true about the following code? (Choose all that apply)

```
1: interface HasVocalCords {
2:   public abstract void makeSound();
3: }
4: public interface CanBark extends HasVocalCords {
5:   public void bark();
6: }
```

A. The `CanBark` interface doesn't compile.

B. A class that implements `HasVocalCords` must override the `makeSound()` method.

C. A class that implements `CanBark` inherits both the `makeSound()` and `bark()` methods.

D. A class that implements `CanBark` only inherits the `bark()` method.

E. An interface cannot extend another interface.

C. The code compiles without issue, so option A is wrong. Option B is incorrect, since an abstract class could implement `HasVocalCords` without the need to override the `makeSound()` method. Option C is correct; any class that implements `CanBark` automatically inherits its methods, as well as any inherited methods defined in the parent interface. Because option C is correct, it follows that option D is incorrect. Finally, an interface can extend multiple interfaces, so option E is incorrect.