**====INTERFACE====**

Public abstract classHonda{ interface IcDonanim { interface DisDonanim {

String marka=”Honda”; koltuk(); direksiyon(); ayna();

String marka=”Mazda”; far(); kaporta(); ayna();

hareket(); dizel(){“dizel”} benzin(){“benzin”}

public class Civic extends Honda implements IcDonanim,Lastik{

public interface Lastik extends DisDonanim{

String marka=”Audi”; garanti(); jant();

İnterface bir class değildir, interface interface dir.

İnterface yi bir class a parent yaparken “implements” keyword ü kullanılır.(tersi , class ın parent ı interface olamaz).

İnterface yi interface ye parent yaparken “extends” keyword ü kullanılır.

Bir class ı bir class a parent yaparken “ extends” keyword ü kullanılır. yukarıdaki şekilde parent child ilişkisi yapılmıştır. Lastik interface i DisDonanim interface nin child ı olduğundan, Civic classs ı Lastik interface sine implements yapıldığında otomatik olarak DisDonanim interface ine de child olur.

İnterface lerde “concrete” metodlar kullanılamaz, sadece abstract metodlar kullanılabilir. İnterface ler SADECE mecbur yapılması gerekenleri depolamak için kullanılır. Yani şekle göre Civic class ında, hareket(), koltuk(), direksiyon(), far(), kaporta(),ayna() ve isteğe göre dizel() metodları bulunur.

Java, Multiple Inheritance ı desteklemez ama parent lar class ise desteklemez, paren lar interface ise Java Multiple Inheritance ı destekler.

Birden fazla parent class olduğunda farklı parent class larda aynı isimli farklı body li metodların olması mümkündür. Bu tip durumlarda child class, aynı isimli farklı body li metodlardan hangisini kullanacağına karar veremez. Fakat interface lerde metodların body leri olmadığından aynı isimli metodlar birbirinin aynısıdır. Bu metodlardan birini override etmek hepsini override etmek manasına gelir.(mesela anne ve baba da ye(); metodu var, çocuk ye(); metodunu seçtiğinde anne de baba da mutlu olur. ).

Not= parent interface ler içinde, aynı metod signature a sahip ama return type ları farklı metodlar varsa Java interface ler içinde CTE verir. Mesela Anne interface de int ye(); Baba interface de void ye(); metodları varsa…

Java interface ler içindeki tüm metodların abstract metod olması gerektiğini bilir. Bu yüzden siz metod üretirken abstract yazsanız da olur yazmasanız da olur. ”public abstract void koltuk();” ile “public void koltuk();” tamamıyla aynıdır.

İnterface lerdeki herşey otomatik olarak “public” dir. Bu yüzden Access modifier ı yazsanız da olur yazmasanız da olur. ”public abstract void koltuk();” ve “public void koltuk();” ve “void koltuk(); tamamıyla aynıdır. Normalde access modifier yazmayınca Access modifier “default” olur ama interface lerde Access modifier her zaman “public” olduğundan yazmasanız da o public dir.

İnterface deki tüm variable lar otomatik olarak “final” olurlar ve final variable ların değeri değiştirilemez.. final variable oluşturduğunuzda isimleri “bold italic” olarak yazılır. (----🡪 int ***FİYAT***=20000; ). Mesela yukarıdaki şekle göre; Civic c1=new Civic(); objesi oluşturduk. c1.***FİYAT=***30000; CTE verir.

Şekildeki fiyat variable ına interface ismi ile ulaşabiliriz. Object üretmeye gerek yok. Demek ki interface deki variable lar static dir. örneğin direk🡪 IcDonanım.fiyat yazarak İç donanım interface inden variable çağırabiliriz.

İnterface deki tüm class member lar “public”dir. öyle ise, interface deki variable lar public+ static + final dir.

public static final int ***FİYAT***=20000; ile int ***FİYAT***=20000; tamamıyla aynıdır.

Aynı isimli aynı data type li variable lar parent interface lerde oluşturulabilir. Ama aynı isimli variable ları object kullanarak çağırırsanız, Java hangisini seçeceğine karar veremez ve aşağıdaki gibi CTE verir. Sout(c1.MARKA);--🡪CTE. CTE almamak için aynı isimli variable lar kesinlikle interface isimleri kullanılarak çağrılmalıdır. Zaten interface lerdeki tüm variable lar static olduğundan doğrusu variable ları interface ile çağırmaktır. Java static leri object ile çağırınca sarı alt çizgi verir.- 🡪sout(IcDonanim.marka); //Honda 🡪sout(DısDonanim.marka);//Mazda 🡪sout(Lastik.marka);//Audi

Metod oluştururken “default” veya “static” keyword lerini kullanarak isterseniz interface içinde “concrete metod” oluşturabilirsiniz. !!! interface içinde concrete metod oluşturmak için kullanılan “default” keyword ü bildiğimiz Access modifier default değildir. Aşağıdaki metodda gördüğünüz gibi Access modifier “public” dir. böyle oluşturulan metodlara “default metod” denir. 🡪public default int add() { return 2;}

Aşağıda gördüğünüz gibi interface içinde concrete metod oluşturmanın 2 yolu var; 1) “default” keyword ünü kullanmak 🡺 Başka class lardan bu metoda ulaşmak için object oluşturmalısınız. 2)”static” keyword ünü kullanmak 🡺 Başka class lardan bu metoda ulaşmak için object oluşturmanıza gerek yok, sadece interface ismi ile metoda ulaşabilirsiniz. Mesela;

public interface I01; …………………. İnterface parent olsun

public default int add() { return2; }

public static int multiply() {return 12; }………interface e bu iki metodu yazalım

public class Runner{……..main metodlu Runner class ından yukarıdaki metodları çağıralım

sout(I01.multiply()); //12 …………multiply() static keyword ü ile oluşturulduğundan ulaşmak için sadece interface ismi yeterlidir.

default olan add metoduna ulaşmak için I01 den obje üretmeliyiz, object üretmeden ulaşmanız mümkün değildir. ama interface den obje üretilmez, bu yüzden bu interface e uzanan bir class oluşturmalıyım.

public class Runner implements I01{…..yukarıdaki örnekteki class ı I01 in child ı yaptım.

Runner obj=new Runner();……..obje oluşturduk

Sout(obj.add());//2 ………..default olan add() metoduna ulaştık.

Concrete bir class, bir interface in child class ı olduğunda interface deki tüm abstract metodları override etmek zorundadır. Fakat bu zorunluluk “default” ve “static” keyword ler kullanılarak oluşturulan concrete metodlar için yoktur.

Yukarıdaki örnekte gördüğümüz gibi, add() ve multiply() metodlarını override etmedik çünkü öyle bir zorunluluk yok ama istersek yaparız.