TECHPROED

PROFESSIONAL TECHNOLOGY EDUCATION

WELCOME TO TECHPROED JAVA TUTORIAL

Testi baslatmak icin asagidaki adımları takip ediniz

Go to www.socrative.com

Click on Login

Click on Student Login

***Room Name*: A L P T E K I N 3 5 2 3**

Kayıtta kullandığınız ismi tam olarak yazınız

Time: 11 Minutes

TECHPROED
PROFESSIONAL TECHNOLOGY EDUCATION

Interface

Interface class'a benzer ama class degildir.

Interface'e sadece abstract methods konabilir.

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

Interface icindeki variable'lari mutlaka initialize etmek zorundasınız,
aksi takdirde Compile Time Error alirsiniz. *int a = 12;* gibi yapmalisiniz.

```
interface Example {
      public void add(){ };   // methods abstract olmali halbuki add() concrete
      public static final int num1 = 12;  // variablelar public static ve final
                                          // initialize da edilmis, problem yok
}
```

```
interface Example {
      void add();   // methods are public by default
      int num1 = 12;  // Variables are public static and final by default
}
```

```
interface Example {
      int num1;   // Compile Time Error
      int num1 = 12;  // It is okay because it is initialized
}
```

Note: public void add(); ve void add(); interface icin aynidir

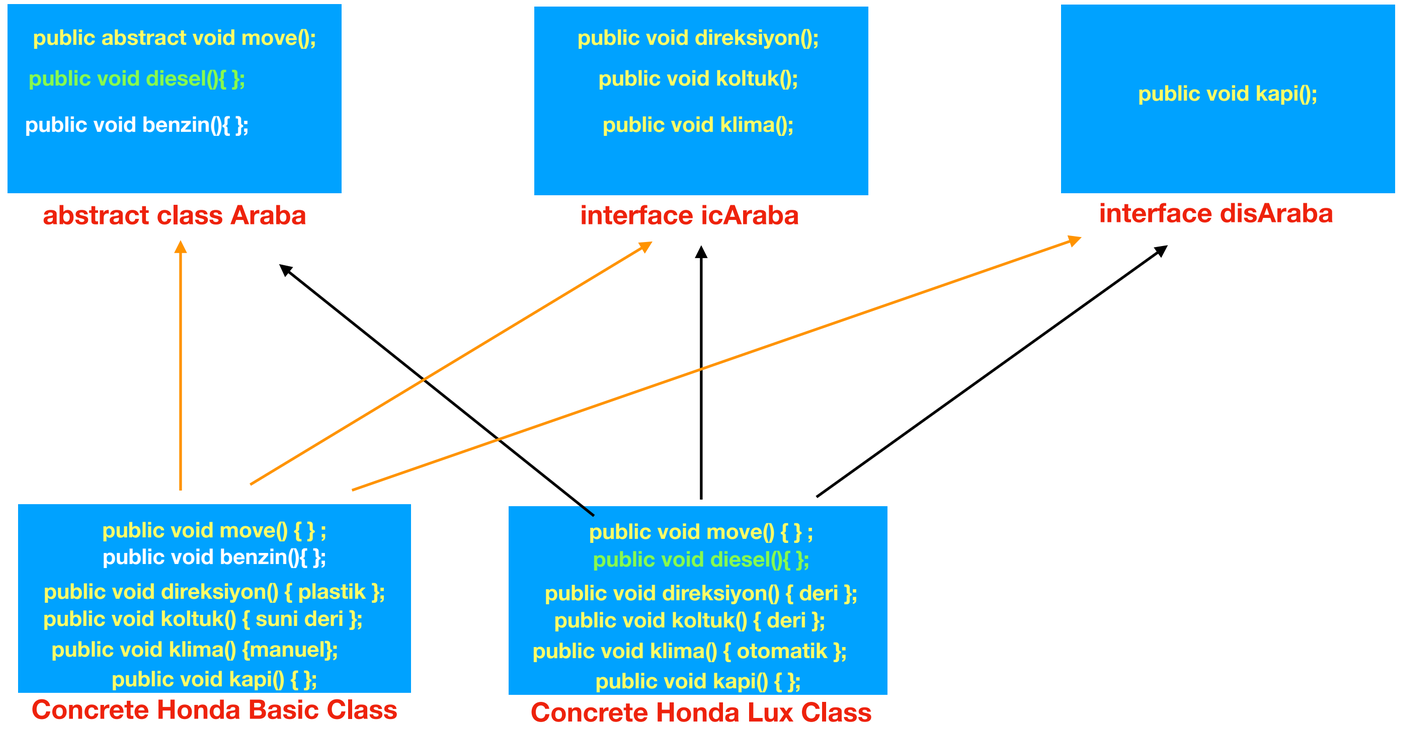
Note: public static final int num1 = 12; ve int num1 = 12; interface icin aynidir

Design uzerinde calisin...

4 tip calculator var

1) Simple Calculator (+, -, :, x, %)
2) Advanced Calculator (Simple Calculator + Denklem Cozme, Karekok, Uslu Sayilar)
3) Scientific Calculator (Advanced Calculator + Logaritma, Trigonometri, Integral)
4) Graphic Calculator (Scientific + Graphic çizebiliyor)

Interface Project



TECHPROED
PROFESSIONAL TECHNOLOGY EDUCATION

“interface” e niçin ihtiyac duyarız?

1: Concrete class ları interface’deki tum metodları implement etmek zorunda bırakır.

2: Abstract class kullandigimizda multiple inheritance mümkün olmaz ama interface’de multiple inheritance mümkün

```
interface Animal {
    void eat ( ) ;
    void drink ( ) ;
}

interface Mammal {
    void nurse ( ) ;
    void blood ( ) ;
    void heart ( );
}

public class Cat implements Animal, Mammal {
    void eat ( ) {
        System.out.println(“Cats eat”);
    }
    void drink ( ) {
        System.out.println(“Cats drink”);
    }
    void nurse ( ) {
        System.out.println(“Cats nurse their young with milk”);
    }

    void blood ( ) {
        System.out.println(“Cats are warm-blooded”);
    }

    void heart ( ) {
        System.out.println(“Cats possess four-chambered hearts”);
    }

}
```

```
interface Animal {
    void eat ( ) ;
    void drink ( ) ;
}

interface Mammal {
    void nurse ( ) ;
    void blood ( ) ;
    void heart ( );
}

Public abstract class Cat implements Animal, Mammal {

}
```

Note: Abstract class lar interface deki abstract metodları implement etmek zorunda degildirler

TECHPROED
PROFESSIONAL TECHNOLOGY EDUCATION

Key Points

1: Interface'den object oluşturulamaz.

2: Interface tum methodlar abstract olmak zorundadır.
Ancak abstract class'lar abstract ve concrete methodlar icerebilir.

3: Class ==> interface : implements
Class ==> Class : extends
Interface ==> interface : extends

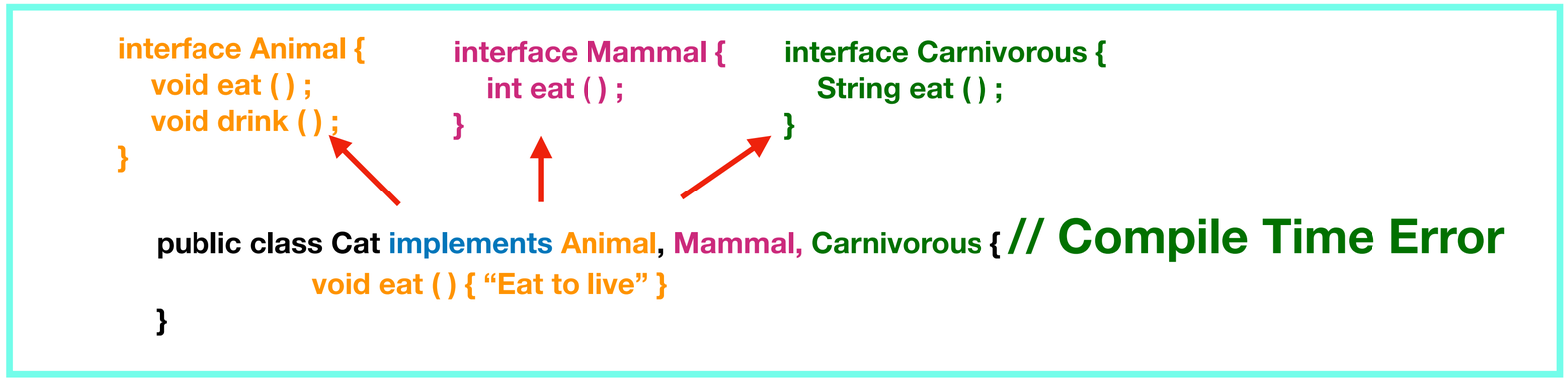
4: Bir class birçok interface'i implement edebilir.

5: Parent interfacelerde ayni isme ve return type'a sahip birçok method varsa,
o methodlardan sadece birini implement etmek yeterlidir

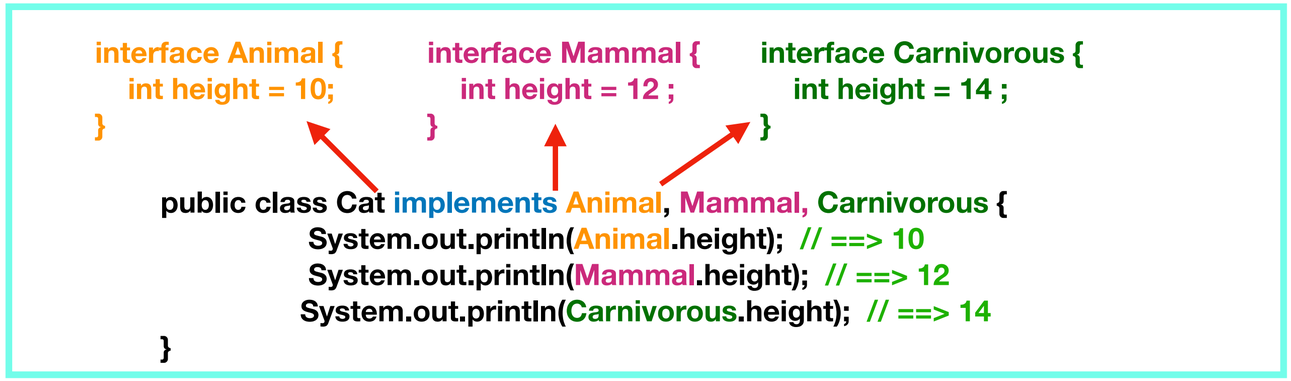
```
interface Animal {
    void eat ( ) ;
    void drink ( ) ;
}

interface Mammal {
    void eat ( ) ;
}

interface Carnivorous {
    void eat ( ) ;
}

public class Cat implements Animal, Mammal, Carnivorous {
        void eat ( ) { "Eat to live" }
}
```

TECHPROED
PROFESSIONAL TECHNOLOGY EDUCATION

5: Bir class ayni isme farklı return type'a sahip metodları implement edemez



6: Ayni isme sahip variable'lar kullanilabilir, istenen variable interface ismi kullanılarak cagrilabilir.



7:“default” or “static” keywordler kullanılarak interface icinde concrete methodlar oluşturabilir

```
public interface Cat {

    public default int drink ( ) {
        return 20;
    }

    public default void eat ( ) {
        System.out.println(“Cats eat”);
    }

    public default  int drink ( );  // Compile Time Error

    public void eat ( ) {                    // Compile Time Error
        System.out.println(“Cats eat”);
    }

}
```

```
public interface Cat {

    public static int drink ( ) {
        return 20;
    }

    public static void eat ( ) {
        System.out.println(“Cats eat”);
    }

    public static int drink ( );  // Compile Time Error

    public void eat ( ) {                    // Compile Time Error
        System.out.println(“Cats eat”);
    }

}
```

8: Bir interface public veya default access modifier'lar kullanabilir ama protected veya private kullanamaz

```
public interface Cat {

}
```

```
interface Cat {

}
```

```
private interface Cat {

}
```

```
protected interface Cat {

}
```

Project for Abstract Class and Interface

```
public abstract class Car {
    public abstract void move();
    public void Diesel(){
      “This car works with diesel”
    }
    public void Gas(){
      “This car works with gas”
    }
}
```

```
public interface Inside {
    public abstract void seat();
    public abstract void steeringWheel();
    public abstract void engine();
}
```

```
public interface Outside {
    public abstract void tire();
    public abstract void door();
    public abstract void window();
}
```

```
public class HondaCivic extends Car implements Inside, Outside {
    public void move(){
        “This car can move fast in sport mode”
    }
    public void Gas(){
      “This car works with gas”
    }
    public void seat(){
      “This car has 5 seats”
    }
    public void steeringWheel(){
       “This car has leather Steering Wheel”
    }
    public void engine(){
        “This car has 1.8 VTI engine”
    }
    public void tire(){
      “This car has 16 inches tires”
    }
    public void door(){
       “This car has automatic door”
    }
    public void window(){
        “This car has colorful windows”
    }
}
```

Questions

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`

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.

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.

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.

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.

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.

Answers

1) 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.

2) 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.

3) 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.

4) 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.

5) 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.

6) 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.