IT 179
4
- Abstract Classes & Methods
- Interfaces

- Let us revisit the Animal Farm example …

`Animal` Polymorphism Example

- Bad design!
- We will never need to create a **`new Animal()`**.
- What is an Animal really?
- Class Animal describes general characteristics that subclasses will inherit.

Abstract Classes

- No need to implement specific methods for `class Animal` as `Animal` should never be instantiated.
- Do this by declaring the class `abstract`

```
abstract class Animal
        {
            public void sleep() ;
            {
                System.out.println("The animals are sleeping");
            }
        }
```

Abstract Classes

- Why would you declare a class abstract if you are not going to instantiate it?

- We need this for **polymorphism**

- Animal Farm Example
  - Need the class Animal for polymorphism
  - But we do not need to write the bodies of the method **sleep()**

Abstract Classes

- To use an abstract class, it must be subclassed and its abstract methods overridden with methods that implement a body.
- In other words, somewhere along the inheritance chain all abstract methods must be implemented.

Abstract Classes

```
abstract class A {
    abstract void m1();
    abstract void m2();
    abstract void m3();
}

class B extends  A {
    void m1() {}
    void m2() {}
    void m3() {}
}
```

Abstract Classes

```
abstract class A {
    abstract void m1();
    abstract void m2();
    abstract void m3();
}

class B extends  A {
    void m1() {}
    void m2() {}
 }
```

Abstract Classes

```
abstract class A {
    abstract void m1();
    abstract void m2();
    abstract void m3();
}

abstract class B extends  A {
    void m1() {}
    void m2() {}
 }
```

Abstract Classes

```
abstract class A {
    abstract void m1();
    abstract void m2();
    abstract void m3();
}

abstract class B extends  A {
    void m1() {}
    void m2() {}
 }

class C extends B {

}
```

Abstract Classes

```
abstract class A {
    abstract void m1();
    abstract void m2();
    abstract void m3();
}

abstract class B extends  A {
    void m1() {}
    void m2() {}
 }

class C extends B {
    void m3() {}
}
```

Abstract Classes


```
abstract class A
{
    abstract void m1();
    abstract void m2();
    abstract void m3();
}

abstract class B extends A
{
    void m1()   {}
}

abstract class C extends B
{
    void m2()   {}
}


abstract class D extends C
{
    void m3()   {}
}

class E extends D {

}
```

Abstract Classes

- Subclasses do not need to implement all abstract methods

- But if a single method is still not implemented, the class must be declared `abstract`.

What's wrong with this?


```
abstract class A
{
    abstract void m1();
    abstract void m2();
    abstract void m3();
}

abstract class B extends A
{
    void m1()   {}
}

class C extends B
{
    void m2()   {}
}
```

Fix 1


```
abstract class A
{
    abstract void m1();
    abstract void m2();
    abstract void m3();
}

abstract class B extends A
{
    void m1()   {}
}

class C extends B
{
    void m2()   {}
    void m3()   {}
}
```

Fix 2


```
abstract class A
{
    abstract void m1();
    abstract void m2();
    abstract void m3();
}

abstract class B extends A
{
    void m1()   {}
}

abstract class C extends B
{
    void m2()   {}
}
```

Is this valid?

```
abstract class A
{
    abstract void m1();
    abstract void m2();
    abstract void m3();
}

abstract class B extends A
{
    void m1()   {}
}

abstract class C extends B
{
    void m2()   {}
}


abstract class D extends C
{
    void m3()   {}
}
```

Is this valid?

```
abstract class A
{
    abstract void m1();
}

abstract class B extends A
{
    abstract void m2();
    void m1()   {}
}

abstract class C extends B
{
    abstract void m3();
    void m2()   {}
}

class D extends C
{

}
```

Abstract Classes

- Abstract classes and abstract methods provide a framework (template.)
- Subclasses implement the details.

- **Demo:** `com.grape.animal`

IT 179

Interfaces

Interfaces

- Suppose you have the following inheritance

```
abstract Animal
-- abstract Mammal
     -- abstract Canine
          -- Class Dog
          -- Class Wolf
     -- abstract Feline
          -- class Lion
          -- class Cat
```

Interfaces

- Would like to add Pet behaviors to **Dog** and **Cat**, such as `beFriendly()`, `play()`, `beProtective()`, **etc.**
- How would you do it?

```
abstract Animal
-- abstract Mammal
      -- abstract Canine
            -- Class Dog
            -- Class Wolf
      -- abstract Feline
            -- class Lion
            -- class Cat
```

One Way

- Just add methods to the Dog and Cat classes.

```
class Dog extends Canine {
    void beFriendly() {}
    void play() {}
    void beProtective() {}
}

class Cat extends Feline {
    void beFriendly() {}
    void play() {}
    void beProtective() {}
}
```

Does this work? Why or Why not?

One Way


```
class Dog extends Canine {
    void beFriendly() {}
    void play() {}
    void beProtective() {}
}

class Cat extends Feline {
    void beFriendly() {}
    void play() {}
    void beProtective() {}
}
```

- Pros: Dog and Cat get behaviors.
- Con: Need to implement multiple times.

Another Way

- add concrete pet methods to the Animal Class?

```
abstract class Animal
{
    void beFriendly() {}
    void play() {}
    void beProtectivce() {}
}
```

- Pros: Dog and Cat get behaviors.
- Con: so does every other class.

Is a Lion a Pet?

- Lion, and every other class would get pet behaviors.
- We can ignore those behaviors, but they are still **part of the Lion class.**

A 3rd Way

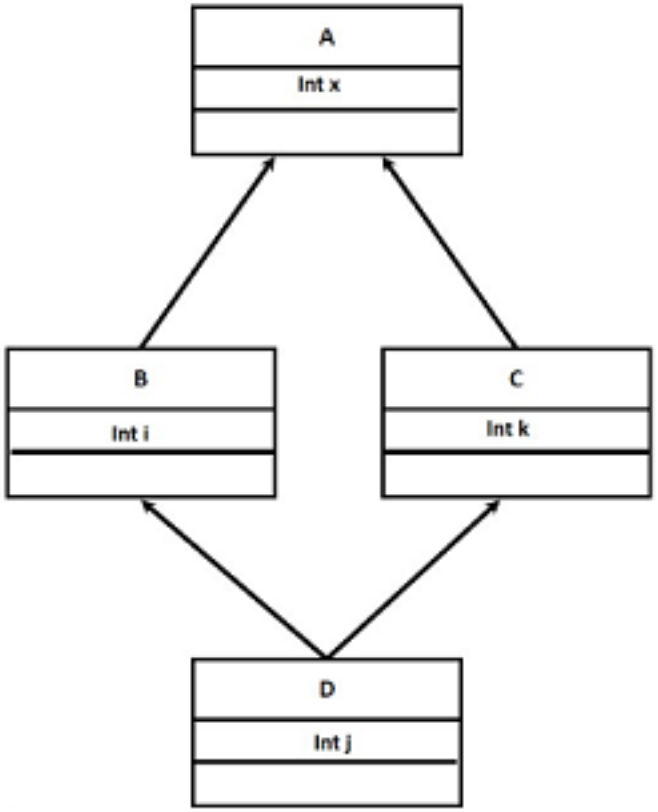
- add abstract pet methods to the Animal Class?

```
abstract class Animal
{
    abstract void beFriendly();
    abstract void play();
    abstract void beProtectivce();
}
```

- Pros: Dog and Cat get behaviors.
- Con: Now every other class needs to implement the pet methods.
- For example, a lion is not a pet but `class Lion` still needs to implement pet methods.

Interfaces

- **Interfaces** solve this problem.
- Java doesn't allow multiple inheritance.
- The **Diamond** of Death Problem.

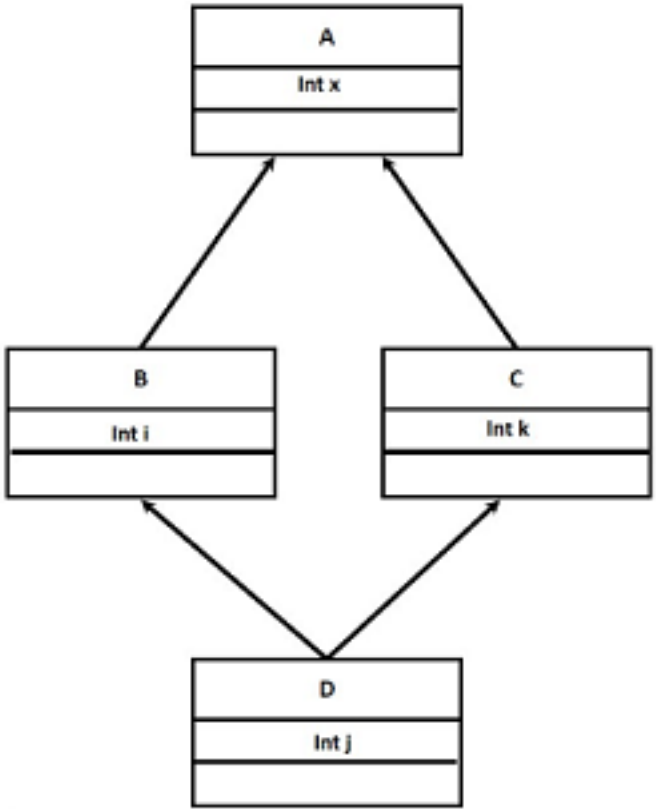


Multiple inheritance (not allowed in Java)

```
class A  {
public void display() {
System.out.println("class A display() method called");
}
}

class B extends A  {
@Override
public void display() {
System.out.println("class B display() method called"); }}

class C extends A  {
@Override
public void display() {
System.out.println("class C display() method called"); }}


//not supported in Java
public class D extends B,C
{
public static void main(String args[])
{
        D d = new D();
        //creates ambiguity which display() method to call
        d.display();
|
}
```

Interfaces

- In Java, you use interfaces to allow "multiple inheritance"
- A class can only extend one class, but **can implement multiple interfaces**.



Interfaces

- Interfaces are abstract classes, but without the keyword `abstract` attached to the methods or class name.
- Instead the class name is declared as an `interface`. Methods have no body.

```
public interface Pet {
    public boolean isFriendly();
    public boolean beFriendly();
    public void play();
}
```

Interfaces - Example

- Demo:
  - `com.grape.petsounds`