|  |  |  |
| --- | --- | --- |
| Main | Animal | Dog |
| **package** com.company;  **public class** Main {   **public static void** main(String[] args) {  Dog dog = **new** Dog(**"jim"**);  dog.eat();  dog.breath();  } }  //an abstract class cant be initiated directly | **package** com.company;  **public abstract class** Animal {  **private** String **name**;   **public abstract void** eat();  **public abstract void** breath();   **public** String getName() {  **return name**;  }   **public void** setName(String name) {  **this**.**name** = name;  } } | **package** com.company;  **public class** Dog **extends** Animal {  @Override  **public void** eat() {  System.***out***.println(getName() + **" is big"**);  }   @Override  **public void** breath() {  System.***out***.println(getName() + **" cant breath, it sucks"**);  }   **public** Dog(String name) {  setName(name);  } } |
| **jim is big**  **jim cant breath, it sucks** | | |

**Adding another class**

|  |  |  |  |
| --- | --- | --- | --- |
| **Main** | **Animal** | **Dog** | **Bird** |
| **package** com.company;  **public class** Main {   **public static void** main(String[] args) {  *//as i had a constructor for Dog class* Dog dog = **new** Dog(**"jim"**);  dog.eat();  dog.breath();   System.***out***.println();  *//bird class  //here i had overridden the Animal's .setName method* Bird bird = **new** Bird();  bird.setName(**"cock"**);  bird.breath();  bird.eat();  } } | **package** com.company;  **public abstract class** Animal {  **private** String **name**;   **public abstract void** eat();  **public abstract void** breath();   **public** String getName() {  **return name**;  }   **public void** setName(String name) {  **this**.**name** = name;  } } | **package** com.company;  **public class** Dog **extends** Animal {  @Override  **public void** eat() {  System.***out***.println(getName() + **" is big"**);  }   @Override  **public void** breath() {  System.***out***.println(getName() + **" cant breath"**);  }   **public** Dog(String name) {  setName(name);  } } | **package** com.company;  **public class** Bird **extends** Animal {   @Override  **public void** eat() {  System.***out***.println(getName()+ **" cant eat"**);  }   @Override  **public void** breath() {  System.***out***.println(getName() + **" can breath"**);  }   @Override  **public void** setName(String name) {  **super**.setName(name);  } } |

**Adding FLY method, since every bird cant fly, better to put it abstract in BIRD.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Main** | **Bird** | **Hen** | **Animal** | **dog** |
| **package** com.company;  **public class** Main {   **public static void** main(String[] args) {  *//as i had a constructor for Dog class* Dog dog = **new** Dog(**"jim"**);  dog.eat();  dog.breath();   System.***out***.println();  *//bird class   //here i had overridden the  // Animal's .setName method   //an abstract class cant be initiated  //directly  /\*Bird bird = new Bird();  bird.setName("cock");  bird.breath();  bird.eat();\*/   //hen class* Hen hen = **new** Hen();  hen.setName(**"hen"**);  hen.breath();  hen.eat();  } } | **package** com.company;  **public abstract class** Bird **extends** Animal {   @Override  **public void** eat() {  System.***out***.println(getName()+ **" cant eat"**);  }   @Override  **public void** breath() {  System.***out***.println(getName() + **" can breath"**);  }   @Override  **public void** setName(String name) {  **super**.setName(name);  }   *//PUTTING AN ABSTRACT CLASS* **public abstract void** fly(); } | **package** com.company;  **public class** Hen **extends** Bird {  @Override  **public void** fly() {  System.***out***.println(getName()+ **" is in my belly"**);  }   @Override  **public void** setName(String name) {  **super**.setName(name);  } } | **package** com.company;  **public abstract class** Animal {  **private** String **name**;   **public abstract void** eat();  **public abstract void** breath();   **public** String getName() {  **return name**;  }   **public void** setName(String name) {  **this**.**name** = name;  } } | **package** com.company;  **public class** Dog **extends** Animal {  @Override  **public void** eat() {  System.***out***.println(getName() + **" is big"**);  }   @Override  **public void** breath() {  System.***out***.println(getName() + **" cant breath, it sucks"**);  }   **public** Dog(String name) {  setName(name);  } } |
| **jim is big**  **jim cant breath, it sucks**  **hen can breath**  **hen cant eat** | | | | |

**Adding an interface that going to give that flexibility Bird class…**

**Because we don’t have a need to implement the fly() method everytime a class that implements Bird (abstract class)**

|  |  |  |
| --- | --- | --- |
| **package** com.company;  **public class** Main {   **public static void** main(String[] args) {  *//as i had a constructor for Dog class* Dog dog = **new** Dog(**"jim"**);  dog.eat();  dog.breath();   System.***out***.println();  *//bird class   //here i had overridden the  // Animal's .setName method   //an abstract class cant be initiated  //directly  /\*Bird bird = new Bird();  bird.setName("cock");  bird.breath();  bird.eat();\*/   //hen class* Hen hen = **new** Hen();  hen.setName(**"hen"**);  hen.breath();  hen.eat();   System.***out***.println();  *//peacock class* Peacock peacock = **new** Peacock();  peacock.setName(**"peacock"**);  peacock.breath();  peacock.eat();  } } | **package** com.company;  **public abstract class** Bird **extends** Animal **implements** canFly {   @Override  **public void** eat() {  System.***out***.println(getName()+ **" cant eat"**);  }   @Override  **public void** breath() {  System.***out***.println(getName() + **" can breath"**);  }   @Override  **public void** setName(String name) {  **super**.setName(name);  }   *//PUTTING AN ABSTRACT CLASS  //public abstract void fly();   //in this way actually we gonna  //giving the flexibility to the Bird class  //so that all classes that is extends Bird  //shouldn't have to have the fly() implementation  //hen was doing so  //we can override fly() any time.* @Override  **public void** fly() {  System.***out***.println(getName() + **" can i fly ? "**);  } } | **package** com.company;  **public class** Hen **extends** Bird {  *//this time if we command this  // codes we'll not get thye error  /\*@Override  public void fly() {  System.out.println(getName()+ " is in my belly");  }\*/* @Override  **public void** setName(String name) {  **super**.setName(name);  } }  **canFly interface**  **package** com.company;  **public interface** canFly {  **void** fly(); } |
| **package** com.company;  **public class** Peacock **extends** Bird {  *//intentionally overriding  //the fly() method..* @Override  **public void** fly() {  **super**.fly();  System.***out***.println(getName() + **" i cant fly"**);  }   *//overriding the setter method  //from thye bird class* @Override  **public void** setName(String name) {  **super**.setName(name);  } } | **package** com.company;  **public abstract class** Animal {  **private** String **name**;   **public abstract void** eat();  **public abstract void** breath();   **public** String getName() {  **return name**;  }   **public void** setName(String name) {  **this**.**name** = name;  } } | **package** com.company;  **public class** Dog **extends** Animal {  @Override  **public void** eat() {  System.***out***.println(getName() + **" is big"**);  }   @Override  **public void** breath() {  System.***out***.println(getName() + **" cant breath, it sucks"**);  }   **public** Dog(String name) {  setName(name);  } } |

**jim is big**

**jim cant breath, it sucks**

**hen can breath**

**hen cant eat**

**peacock can breath**

**peacock cant eat**

**DIFFERENCE BETWEEN ABSTRACT CLASS AND INTERFACE**

|  |  |
| --- | --- |
| enter image description here | 1. Main difference is methods of a Java interface are implicitly abstract and cannot have implementations. A Java abstract class can have instance methods that implements a default behavior. 2. Variables declared in a Java interface is by default final. An  abstract class may contain non-final variables. 3. Members of a Java interface are public by default. A Java abstract class can have the usual flavors of class members like private, protected, etc.. 4. Java interface should be implemented using keyword “implements”; A Java abstract class should be extended using keyword “extends”. 5. An interface can extend another Java interface only, an abstract class can extend another Java class and implement multiple Java interfaces. 6. A Java class can implement multiple interfaces but it can extend only one abstract class. 7. Interface is absolutely abstract and cannot be instantiated; A Java abstract class also cannot be instantiated, but can be invoked if a main() exists. 8. In comparison with java abstract classes, java interfaces are slow as it requires extra indirection. |