|  |  |  |
| --- | --- | --- |
| **MAIN** | **CLASS THAT IMPLEMENTS INTERFACE** | **INTERFACE** |
| **package** com.company;  **public class** Main {   **public static void** main(String[] args) {  grant\_interface gf = **new** grant\_interface();  gf.**name** = **"Soham Ivy"**;  gf.**marks** = 50;  gf.name();  System.***out***.println(**"got marks twice, wah chutiya "** + gf.marks());  } } | **package** com.company;  **public class** grant\_interface **implements** Ibase {   String **name**;  **int marks**;   @Override  **public void** name() {  System.***out***.println(**"your name is "** + **this**.**name**);  }   @Override  **public int** marks() {  **int** m = **this**.**marks** \* 10;  **return** m;  } } | **package** com.company;  *//interface definition //https://www.tutorialspoint.com/java/java\_interfaces.htm* **public interface** Ibase {  **void** name();  **int** marks(); } |

A CLASS CAN INHERIT MULTIPLE INTERFACES

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Main | Interfaces | Dog | Plane | Bird |
| **package** com.company;  **public class** Main {   **public static void** main(String[] args) {  Dog d = **new** Dog();  d.walk();   plane p = **new** plane();  p.fly();   bird b = **new** bird();  b.fly();  b.walk();  } } | **package** com.company; **public interface** Ibase {  **void** walk(); }  **package** com.company;  **public interface** Ibase2 {  **void** fly(); } | **package** com.company;  **public class** Dog **implements** Ibase {   @Override  **public void** walk() {  System.***out***.println(**"Dog barks"**);  } } | **package** com.company;  **public class** plane **implements** Ibase2 {  @Override  **public void** fly() {  System.***out***.println(**"A plane can fly"**);  } } | **package** com.company;  *//multiple inheritance is allowed by a class* **public class** bird **implements** Ibase2, Ibase {  @Override  **public void** walk() {  System.***out***.println(**"Bird can fly"**);  }   @Override  **public void** fly() {  System.***out***.println(**"Bird can walk"**);  } } |

**When overriding methods defined in interfaces, there are several rules to be followed −**

* Checked exceptions should not be declared on implementation methods other than the ones declared by the interface method or subclasses of those declared by the interface method.
* The signature of the interface method and the same return type or subtype should be maintained when overriding the methods.
* An implementation class itself can be abstract and if so, interface methods need not be implemented.

When implementation interfaces, there are several rules −

* A class can implement more than one interface at a time.
* A class can extend only one class, but implement many interfaces.
* An interface can extend another interface, in a similar way as a class can extend another class.

Extending Interfaces

An interface can extend another interface in the same way that a class can extend another class. The **extends** keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

The following Sports interface is extended by Hockey and Football interfaces.

Example

// Filename: Sports.java

public interface Sports {

public void setHomeTeam(String name);

public void setVisitingTeam(String name);

}

// Filename: Football.java

public interface Football extends Sports {

public void homeTeamScored(int points);

public void visitingTeamScored(int points);

public void endOfQuarter(int quarter);

}

// Filename: Hockey.java

public interface Hockey extends Sports {

public void homeGoalScored();

public void visitingGoalScored();

public void endOfPeriod(int period);

public void overtimePeriod(int ot);

}

The Hockey interface has four methods, but it inherits two from Sports; thus, a class that implements Hockey needs to implement all six methods. Similarly, a class that implements Football needs to define the three methods from Football and the two methods from Sports.

Extending Multiple Interfaces

A Java class can only extend one parent class. Multiple inheritance is not allowed. Interfaces are not classes, however, and an interface can extend more than one parent interface.

The extends keyword is used once, and the parent interfaces are declared in a comma-separated list.

For example, if the Hockey interface extended both Sports and Event, it would be declared as −

Example

public interface Hockey extends Sports, Event

Tagging Interfaces

The most common use of extending interfaces occurs when the parent interface does not contain any methods. For example, the MouseListener interface in the java.awt.event package extended java.util.EventListener, which is defined as −

Example

package java.util;

public interface EventListener

{}

An interface with no methods in it is referred to as a **tagging** interface. There are two basic design purposes of tagging interfaces −

**Creates a common parent** − As with the EventListener interface, which is extended by dozens of other interfaces in the Java API, you can use a tagging interface to create a common parent among a group of interfaces. For example, when an interface extends EventListener, the JVM knows that this particular interface is going to be used in an event delegation scenario.

**Adds a data type to a class** − This situation is where the term, tagging comes from. A class that implements a tagging interface does not need to define any methods (since the interface does not have any), but the class becomes an interface type through polymorphism.