**Interfaces and Inheritance**

* **Interfaces**
  + An interface is a reference type, similar to class, that can contain only constants, method signatures and nested types.
  + There are no method bodies.
  + Interfaces cannot be instantiated, they can only be implemented by classes or extended by other interfaces.
  + To use an interface, you write a class that implements the interface.
  + A class can implement more than one interface.
  + An interface defines a protocol of communication between two objects.
  + A class that implement an interface must implement all the methods declared in the interface.
* **Defining an Interface**
  + An interface declaration consists of modifiers, the keyword interface, the interface name, a comma separated list of parent interfaces and the interface body.

*public interface interfaceName extends ParentInterface1, ParentInterface2 {}*

* + **The Interface Body**
    - The interface body contains the method declarations of all the methods included in the interface.
    - All methods declared in an interface are implicitly public. So the public modifier can be omitted.
    - All constant values defined in an interface are implicitly public, static and final. Once again these modifiers can be omitted.
* **Implementing an Interface**
  + A class can implement more than one interface, so the implements keyword is followed by a comma-separated list of the interfaces implemented by the class.
* **Using an Interface as a Type**
  + When you define a new interface, you are defining a new reference data type.
  + If you define a reference variable whose type is an interface, any object assign to it must be an instance of a class that implements the interface.
* **Rewriting Interfaces**

public interface DoItPlus extends DoIt {

boolean didItWork(int i, double x, String s);

}

* + Now the user code can choose the old interface or to upgrade to the new one.
* **Inheritance**
  + A class that’s derived from another class is called sub class(also a derived class, extended class or child class)
  + The class from which the sub class is derived is called a super class (also a base class or parent class).
  + Every class has one and only one direct super class.
  + In the absence of any other explicit super class, every class is implicitly a sub class of object.
  + The top most class is said to be descended from all other classes.
  + A sub class inherit all the members (fields, methods and nested classes) from its super class.
  + Constructors are not members, so they are not inherited by sub classes but the constructor of the super class can be invoked from sub class.
* **Overriding and Hiding Methods**
  + **Instance Methods**
    - An instance method in a sub class with same signature and return type as in instance method in the super class overrides the superclass’s method.
    - An overriding method can also return a subtype of the type returned by the overridden method. This is called covariant return type.
    - When overriding a method, you might want to use @Override annotation that instructs the compiler that you intend to override a method in the super class.
  + **Class Methods**
    - If a sub class defines a class method with a same signature as a class method in the super class, the method in the sub class hides the one in the super class.
    - The distinction between hiding and overriding is important implications.
    - The version of the overridden method that gets invoked is the one in the sub class.
    - The version of the hidden method that gets invoked depends on whether it is invoked from the super class or the sub class.

public class Animal {

public static void testClassMethod() {

System.out.println("The class" + " method in Animal.");

}

public void testInstanceMethod() {

System.out.println("The instance " + " method in Animal.");

}

}

The second class, a subclass of Animal, is called Cat:

public class Cat extends Animal {

public static void testClassMethod() {

System.out.println("The class method" + " in Cat.");

}

public void testInstanceMethod() {

System.out.println("The instance method" + " in Cat.");

}

public static void main(String[] args) {

Cat myCat = new Cat();

Animal myAnimal = myCat;

Animal.testClassMethod();

myAnimal.testInstanceMethod();

}

}

The Cat class overrides the instance method in Animal and hides the class method in Animal. The main method in this class creates an instance of Cat and calls testClassMethod() on the class and testInstanceMethod() on the instance.

The output from this program is as follows:

The class method in Animal.

The instance method in Cat.

* + **Modifiers**
    - The access specifier for an overriding method can allow more but not less, access than the overridden method.
    - For example, a protected instance method in the super class can be made public but not private in sub class.
* **Polymorphism**
  + Polymorphism is the ability of an object to take on many forms.
  + The most common use of polymorphism in oops occurs when a parent class reference is used to refer to a child class object.
  + Any object that can pass more than one IS-A test is considered to be polymorphic.
  + In Java all the java object are polymorphic.
  + Reference variable can be of only one type. Once declared the type of the reference variable cannot be changed.

public class TestBikes {

public static void main(String[] args){

Bicycle bike01, bike02, bike03;

bike01 = new Bicycle(20, 10, 1);

bike02 = new MountainBike(20, 10, 5, "Dual");

bike03 = new RoadBike(40, 20, 8, 23);

bike01.printDescription();

bike02.printDescription();

bike03.printDescription();

}

}

* + The JVM calls the appropriate method for the object that’s referred to in each variable.
  + That does not call the method that’s defined by the variable’s type.
  + This method id referred to as virtual method invocation.
* **Hiding Fields**
  + Within a class, a field has the same name as a field in the super class hides the super class’s field, even if their types are different.
  + Within the sub class, the field in the super class cannot be referenced by its simple name.
  + Instead the field must be accessed through super.
* **Using the keyword Super**
  + **Accessing Super Class Members**
    - If your method overrides one of its super class methods, you can invoke the overridden method through the use of the keyword *super*.
    - You can also use super to refer to a hidden field.
  + **Subclass Constructors**
    - Invocation of super class constructor must be the first line in the sub class constructor.
    - The syntax for calling super class constructor in

*super ()*

*Or*

*super (parameter list)*

* + - If a constructor does not explicitly invoke a super class constructor, the Java Compiler automatically inserts a call to the no-argument constructor of the super class.
* **Object as a Super Class**
  + The object class in java.lang package sit at the top of the class hierarchy tree.
  + Every class you use or write inherits the instance methods of the object.
  + You need not use any of these methods, but if you choose to do, you may need to override them with the code that’s specific to your class.
  + The methods inherited from object classes are
    - *protected Object clone() throws CloneNotSupportedException*
      * Create and returns a copy of this object.
    - *public boolean equals (Object obj)* 
      * Indicates whether some other object is equal to this one.
    - *protected void finalize() throws Throwable*
      * Called by garbage collector on an object when garbage collection determines that there are no more references to the object.
    - *public final Class getClass()* 
      * Returns the runtime class of an object.
    - *public int hashCode()*
      * Returns the hash code value for the object.
    - *public String toString()*
      * Returns a string representation of the object.
  + The notify(), notifyAll() and wait() methods of object all play a part in synchronizing the activities of independently running threads in a program.
    - * *public final void notify()*
      * *public final void notifyAll()*
      * *public final void wait()*
      * *public final void wait (long timeout)*
      * *public final void wait (long timeout, int naos)*
  + **The clone() Method**
    - If a class or one of its super class implements the *Cloneable* interface, you can use clone method to create a copy from an existing object.
    - To create a clone you write as

*aClonableObject.clone()*

* + - If the Cloneable interface not implemented it will throw the clone not supported exception.
    - If you are going to write a clone() method to override the one in object

protected Object clone() throws CloneNotSupportedException

or

public Object clone() throws CloneNotSupportedException

* + **The equals() Method**
    - The equals() method compares two objects for equality and returns true if they are equal.
    - You should always override the equals() method if the identity operator is not appropriate for your classes.
    - If you override equals() you must override hashCode() as well.
  + **The finalize() Method**
    - You can override the finalize() method to do cleanup, such as freeing resources.
    - The finalized method may be called automatically by the system but when it is called or even if it is called is uncertain.
  + **The getClass() Method**
    - You can override the getClass() method.
    - The getClass() method returns the Class object.
    - Which has methods you can use to get information about the class.
    - Such as its name(getSimpleName()), its super class(getSuperClass()) and the interface its implements(getInterface()).
  + The Class class in the java.lang package has a large number of methods.
  + For example, you can test to see if the class is an annotation (isAnnotation()), an interface (isInterface()) or an enum (isEnum()).
  + To see what the object fields are (getFields()).
  + What its methods are (getMethods()).
  + **The hashCode() Method**
    - The value returned by hashCode() is the object’s hash code.
    - Which is the objects memory address in hexadecimal.
    - If you override the equals() method you must override the hashCode() method as well.
  + **The toString() Method**
    - The object’s toString() method returns a string representation of the object.
* **Writing Final Classes and Methods**
  + You can declare some or all of the class’s methods as final.
  + Use the final keyword in a method declaration to indicate that the method cannot be overridden by subclasses.
  + You can also declare entire class as final.
  + A class that’s declared final cannot be subclassed.
* **Abstract Methods and Classes**
  + Abstract class is a class that is declared abstract.
  + It may or may not include the abstract methods.
  + Abstract classes cannot be instantiated but can be subclassed.
  + Abstract method is a method that’s declared without an implementation.

*abstract void moveTo(double deltaX, double deltaY);*

* + If a class includes abstract methods, the class itself must be declared as abstract.
  + When an abstract class is subclassed, the subclass usually provides the implementation for all of the methods in its parent class.
  + However if it does not, the subclass must also be declared as abstract.
  + All of the methods in an interface are implicitly abstract, so the abstract modifier is not used with interface methods.
  + Abstract class may have static fields and static methods, you can use these class members with a class reference.
  + **Abstract Classes versus Interfaces**
    - Unlike interfaces, abstract classes can contain fields that are not static and final and they can contain implemented methods.
    - Such abstract classes are similar to interfaces except that they provide the partial implementation, leaving it to subclasses to complete the implementation.
    - If an abstract class contain only abstract method declarations, it should be declared as interface instead.