1. Log in
2. Unit 5
   - Inheritance (overriding and more)
   - Polymorphism
   - Abstract Classes
   - Interfaces

Oct 16-7:52 AM

General Ideas behind Abstract Classes/Methods ...

```
public abstract class class1Name{

   /*  Instance Variables  */

   public abstract returnType method1Name( );
   public abstract returnType method2Name( );
   ...

   /*  Other Methods  */
}
```

```
public class class2Name extends class1Name{
   /*  Instance Variables  */
   /*  Constructor Method  */
   public returnType method1Name( ){
        /*  Code to implement method1  */
   }
   public abstract returnType method2Name( ){
        /*  Code to implement method2  */
   }
   /*  Other Methods  */
}
```

Nov 18-6:50 PM

Abstract Classes ...

1. Typically is not an object itself, but a concept.
   Account is a concept ... checking/savings are objects.
2. Typically is never created, but subclasses are.
   Account isn't really created ... checking/savings are.
3. Do not technically have to have a constructor, subclass can.

Abstract Methods...

1. Methods with no code, just a place-holder to be over-written (overridden) in a subclass.
2. MUST be defined in a subclass or errors will occur.
3. Do not use braces (no code), simply ends with a semicolon.

Nov 18-10:22 AM

Previous Lesson ...

*** Account Class**

- Review each item ...
- NOTE: getAccount Number( )
- NOTE: enterAccountInfo( )
- NOTE: earnInterest( )

* All abstract methods need to be defined in a Subclass or you will run into errors!

```
public abstract class Account {

   int number;
   String name;
   String password;

   public Account( ){
      number=0;
      name=" ";
      password="XYZ123";
   }

public abstract void enterAccountInfo(...parameters...);
public abstract int getAccountNumber( );
public abstract void earnInterest( );

public String getPassword(){
      return this.password;
   }
public String getName(){
      return this.name;
   }
}
```

Nov 13-12:26 PM

Interface

A collection (or list) of method headers that will be implemented by another class (organizational tool?)

Nov 19-12:16 PM

Interface

A collection (or list) of method headers that will be implemented by another class (organizational tool?)

Example:

1. I am going to create a class to represent all modes of transportation. This is not an object, but merely represents a bunch of objects.

Nov 19-12:16 PM

Interface

A collection (or list) of method headers that will be implemented by another class (organizational tool?)

Example:

1. I am going to create a class to represent all modes of transportation. This is not an object, but merely represents a bunch of objects.
2. I would like to use an interface that will list all of the methods that each transportation object will implement.

Nov 19-12:16 PM

Interface

A collection (or list) of method headers that will be implemented by another class (organizational tool?)

Example:

1. I am going to create a class to represent all modes of transportation. This is not an object, but merely represents a bunch of objects.
2. I would like to use an interface that will list all of the methods that each transportation object will implement.

```
public interface ModeOfTransportation {
    public abstract boolean isMoving( );
    public abstract void makeMove( );
    public abstract void makeStop( );
    public abstract int getWheels( );
}
```

****
Header uses interface instead of class
****

Nov 19-12:16 PM

Interface

A collection (or list) of method headers that will be implemented by another class (organizational tool?)

Example:

1. I am going to create a class to represent all modes of transportation. This is not an object, but merely represents a bunch of objects.
2. I would like to use an interface that will list all of the methods that each transportation object will implement.

```
public interface ModeOfTransportation {
    public abstract boolean isMoving( );
    public abstract void makeMove( );
    public abstract void makeStop( );
    public abstract int getWheels( );
}
```

****
Any class that implements this interface MUST define all 4 of these!
****

Nov 19-12:16 PM

Interface

Actually, "public abstract" is automatically implied in an interface so it is no longer used/needed ...

```
public interface ModeOfTransportation {
    boolean isMoving( );
    void makeMove( );
    void makeStop( );
    int getWheels( );
}
```

Nov 19-12:16 PM

Writing a Skateboard Class to implement ModeOfTransportation

```
public interface ModeOfTransportation {
    boolean isMoving( );
    void makeMove( );
    void makeStop( );
    int getWheels( );
}
```

****
Any class that implements this interface MUST define all 4 of these!
****

```
public class Skateboard implements ModeOfTransportation{
    public boolean moving;
    public int numberOfWheels;

    public Skateboard(){
        moving=false;
        numberOfWheels=4;
    }

    public boolean isMoving(){
        return this.moving;
    }
    public void makeMove(){
        System.out.println("The Skateboard is now moving ...");
        this.moving=true;
    }
    public void makeStop(){
        System.out.println("The Skateboard is now stopped ...");
        this.moving=false;
    }
    public int getWheels(){
        return this.numberOfWheels;
    }
}
```

Nov 19-12:16 PM

Walk Through the Skateboard Class

* Use the word 'implements' to signify that you are using the interface ModeOfTransportation

```
public class Skateboard implements ModeOfTransportation{
    public boolean moving;
    public int numberOfWheels;

    public Skateboard(){
        moving=false;
        numberOfWheels=4;
    }

    public boolean isMoving(){
        return this.moving;
    }
    public void makeMove(){
        System.out.println("The Skateboard is now moving ...");
        this.moving=true;
    }
    public void makeStop(){
        System.out.println("The Skateboard is now stopped ...");
        this.moving=false;
    }
    public int getWheels(){
        return this.numberOfWheels;
    }
}
```

Nov 19-12:16 PM

Walk Through the Skateboard Class

* Use the word 'implements' →
* Instance variables are used in the Skateboard class, they are never used in an interface

```
public class Skateboard implements ModeOfTransportation{
    public boolean moving;
    public int numberOfWheels;

    public Skateboard(){
        moving=false;
        numberOfWheels=4;
    }

    public boolean isMoving(){
        return this.moving;
    }
    public void makeMove(){
        System.out.println("The Skateboard is now moving ...");
        this.moving=true;
    }
    public void makeStop(){
        System.out.println("The Skateboard is now stopped ...");
        this.moving=false;
    }
    public int getWheels(){
        return this.numberOfWheels;
    }
}
```

Nov 19-12:16 PM

Walk Through the Skateboard Class

* Use the word 'implements' →
* Instance variables →
* Constructor Method used to create instances of Skateboard. Interfaces never create instances

```
public class Skateboard implements ModeOfTransportation{
    public boolean moving;
    public int numberOfWheels;

    public Skateboard(){
        moving=false;
        numberOfWheels=4;
    }

    public boolean isMoving(){
        return this.moving;
    }
    public void makeMove(){
        System.out.println("The Skateboard is now moving ...");
        this.moving=true;
    }
    public void makeStop(){
        System.out.println("The Skateboard is now stopped ...");
        this.moving=false;
    }
    public int getWheels(){
        return this.numberOfWheels;
    }
}
```

Nov 19-12:16 PM

Walk Through the Skateboard Class

* Use the word 'implements' →
* Instance variables →
* Constructor Method →
* The 4 "placeholder methods" from the interface get defined.

```
public class Skateboard implements ModeOfTransportation{
    public boolean moving;
    public int numberOfWheels;

    public Skateboard(){
        moving=false;
        numberOfWheels=4;
    }

    public boolean isMoving(){
        return this.moving;
    }
    public void makeMove(){
        System.out.println("The Skateboard is now moving ...");
        this.moving=true;
    }
    public void makeStop(){
        System.out.println("The Skateboard is now stopped ...");
        this.moving=false;
    }
    public int getWheels(){
        return this.numberOfWheels;
    }
}
```

Nov 19-12:16 PM

Walk Through the Skateboard Class

* Use the word 'implements' →
* Instance variables →
* Constructor Method →
* Interface methods defined
* This class can have other methods, it just doesn't

```
public class Skateboard implements ModeOfTransportation{
    public boolean moving;
    public int numberOfWheels;

    public Skateboard(){
        moving=false;
        numberOfWheels=4;
    }

    public boolean isMoving(){
        return this.moving;
    }
    public void makeMove(){
        System.out.println("The Skateboard is now moving ...");
        this.moving=true;
    }
    public void makeStop(){
        System.out.println("The Skateboard is now stopped ...");
        this.moving=false;
    }
    public int getWheels(){
        return this.numberOfWheels;
    }
}
```

Nov 19-12:16 PM

Analyze a main method for Skateboard and ModeOfTransportation

```
public static void main(String[] args) {
    System.out.println("\n");
    Skateboard skate1 = new Skateboard();

    System.out.println("Here's my first skateboard ...");
    System.out.println("It has "+skate1.numberOfWheels+" wheels.");
    System.out.println("Try this another way ... accessing # of wheels ...");
    System.out.println(skate1.getWheels());    //print since return value
    System.out.println("");

    System.out.println("It is moving ... "+skate1.moving);
    System.out.println("Test to see if it is moving ...");
    System.out.println(skate1.isMoving());     //print with return value
    System.out.println("");

    System.out.println("I'm going to make it move now ...");
    skate1.makeMove();                         //mutator, no return
    System.out.println("See ...");
    System.out.println(skate1.isMoving());     //return value = print
    System.out.println("");

    System.out.println("Now I'll make the skateboard stop ...");
    skate1.makeStop();                         //mutator, no return
    System.out.println("Moving ...");
    System.out.println(skate1.isMoving());     //print return value

}
```

Nov 19-12:16 PM

Things to do ...

1. Wrap Up Unit 5 WS 01-05
2. Work on Unit 5 WS06 Interfaces

Oct 16-9:12 AM