Noah Parker
Chapter 10 Study Sheet

Using Interfaces for Algorithm Reuse like a class interfaces can have methods and variables, but the methods declared are by default abstract. Rules of an abstract method 1. Abstract methods don't have body, they just have method signature as shown above.
2. If a class has an abstract method, it should be declared abstract, the vice versa is not true, which means an abstract class doesn't need to have an abstract method compulsory. 3. If a regular class extends an abstract class, then the class must have to implement all the abstract methods of abstract parent class, or it must be declared abstract as well. Interfaces are useful Java does not support "multiple inheritance" (a class can only inherit from one superclass). However, it can be achieved with interfaces, because the class can implement multiple interfaces. Note: To implement multiple interfaces, separate them with a comma. An example is → unlike inheritances in chapter 9 we need to use the implements keyword instead of extends. As you see in the interface for car it has methods with no body's just like abstract methods.

```
Interface car {
 public void carhonk(); // interface method (does not have a body)
 public void reveEngine(); // interface method (does not have a body)
}
class car implements vehiclel {
 public void carhonk() {
  // The body of carhoke() is provided here
  System.out.println("The car honks: beep beep");
 }
 public void reveEngine() {
  // The body of reveEngine() is provided here
  System.out.println("voom voom");
 }
}
class Main {
 public static void main(String[] args) {
  car mycar = new car();  // Create a car object
  mycar.carhonk();
  mycar.reveEngine();
 }
}
```

The Comparable Interface is used to order the objects of the user-defined class. public int compareTo(Object obj): It is used to compare the current object with the specified object. It returns. An example of it is

```
class car implements Comparable<car>{
int Serialno;
String Brand;
String Make;
Student(int Serialno,String Brand,String Make){
this.Serialno=Serialno;
this.Brand=Brand;
this.Make=Make;
}

public int compareTo(car st){
if(Serialno==st.SerialNo)
return 0;
else if((Serialno<B st.Serialno)
return 1;
else
return -1;
}
}
```

In this example we are comparing brands. When we call on this it will compare if the serial number is greater or less than if the serial number matches it returns 0 but if the serial number is less than the other it returns one and if neither of those it returns a negative 1. This helps us sort the cars by the lowest serial numbers first and print the brand of them and the make of them.

Inner Classes it is possible in java to nest classes which is a class within a class. The purpose of this is to group classes that belong together which can make code more readable and maintainable. To access the inner class, create an object of the outer class, and then create an object of the inner class: an example of this is →

```
class OuterClass {
 String  x = "Hello ";

 class InnerClass {
  String y = "Student";
 }
}

public class Main {
 public static void main(String[] args) {
  OuterClass myOuter = new OuterClass();
  OuterClass.InnerClass myInner = myOuter.new
InnerClass();
  System.out.println(myInner.y + myOuter.x);
 }
}
// Outputs "Hello Student" (Hello + Student)
```

NOTE: WE CAN NOT HAVE A STATIC METHOD IN A NESTED INNER CLASS!!!

Mouse Events The MouseListener interface is found in java.awt.event package. It has five methods which are listed below

Methods of MouseListener interface
public abstract void mouseClicked(MouseEvent e);
public abstract void mouseEntered(MouseEvent e);
public abstract void mouseExited(MouseEvent e);
public abstract void mousePressed(MouseEvent e);
public abstract void mouseReleased(MouseEvent e);

The mousePressed and mouseReleased methods are called whenever a mouse button is pressed or released. If a button is pressed and released in quick succession, and the mouse has not moved, then the mouseClicked method is called as well. The mouseEntered
and mouseExited methods can be used to paint a user-interface component in a special way whenever the mouse is pointing inside it.