Computer Science AP
Interfaces Worksheet

Questions for Videos from the Inheritance Interfaces section.

What is an interface?
The simplest use of an interface is to provide a list of methods that a class can implement (code). A class that codes the methods of an interface is allowed to state that they 'implement' the interface.

What are some similarities between interfaces and abstract classes?
Abstract classes also have methods that are left uncoded.
They can both have variables (but variables in interfaces is not part of AP Course).
Neither can be instantiated (cannot use the *new* keyword to create an instance of one).

What are some differences between interfaces and abstract classes?
Abstract class have methods that are fully coded (interfaces can actually code default methods but it is not part of the AP course).

Can you instantiate an instance of an interface?
(Example: Comparable C = new Comparable())
No. Since it is not fully coded, you cannot create one. Classes are meant to implement an interface. You are not meant to create an instance of one.

What must a class do to be able to 'implement' an interface?
Code all the methods listed in the interface.

Can a class implement more than one interface? If so, how? If no, why not.
Yes. Just use a comma (,) . Example: Student implements Comparable, Gradable, Expellable

What are two popular interfaces that you are required to know about in this course?
List interface and Comparable interface are popular and testable.

For each of the two popular interfaces you named, name a common class that implements the interface.
ArrayList implements List. String implements Comparable.

Would these two lines of code compile and run in a program? Explain.
Comparable temp = new String("Hi")
List myList = new ArrayList<String>()

Why is the method declaration
public void stuff(List<String> S) a better choice than *public void stuff(ArrayList*<String> S) ?

Leaving the parameter as List means that you could send in any type of List into the method. You could send it an ArrayList of Strings, a LinkedList of Strings, a VectorList of Strings, etc. It make the method more flexible and more useable.

Consider the InheritanceExamples project, studentTeacher01 package.
Add an interface to the project called *Mailable* that has the methods
String getName() and Address getAddress() .

Now make changes to the Person class so that it implements the *Mailable* interface.
Go code the two methods in! Make sure they have the same return type, parameters, etc.

When you've checked that there are no compiler errors, copy and paste the code for the *Mailabe* interface and the Person class into this document.

Assuming that your code works as intended, would the following lines of code compile fine?
Mailable temp = new Person("Jenn", 18)
Mailable temp2 = new Student("Bob", 15, 123)

Yes. Since a Person implements Mailable, it qualifies as a reference variable of type Mailable.
Yes. The student class extends Person, so it inherits the interface as well since the methods are coded. See how you can have many classes qualify as a Mailable reference.

If you have a list created with the line List<Mailable> people = new ArrayList<Mailable>() , could you add a Person or a Student into this list?
Yes. Person and Student are both Mailable's.

Consider the InheritanceExamples project, studentTeacher01 package.
Make the Student class implement the Comparable interface using the student's studentId for comparison. A 'larger' studentId will result in a 'larger' Student in the comparison.

```
public int compareTo(Object O) {
   if ( studentId > ((Student)O).studentId)
      return(1);
   else if ( studentId < ((Student)O).studentId)
      return(-1);
   else
      return(0);
}
```

or just the line

```
    return(  studentId – ((Student)O).studentId) ;
```

*You don't have to send back exactly 1 or -1, as long as you send back larger than 0, smaller than 0, or 0 you are fine!

After you have made the Student implement Comparable, use one of the runner programs and compare two Students using the compareTo method. If it works, provide your code here. document.

```
Student Bob = new Student("Bob",  15, 12345);
Student Jane = new Student("Jane", 18, 898989);
if (Jane.compareTo(Bob) > 0 ) {
    System.out.println("Jane is older ("bigger") );
}
```