AP Computer Science   Name: ______________________________
Karel Chapter 4 Review Notes (7)

INTERFACES

Interfaces:

- A Java interface is similar to an abstract class: it has one or several abstract methods declared but left undefined. The difference is that **all** of an interface's methods are abstract: they have headers but not method bodies. Interfaces have no constructors and no other code (except, perhaps, a few static constants).
- The keywords `public abstract` are omitted in the method headers in an interface because every method is public and abstract.
- Once an interface is defined, we can "officially" state that a class `implements` that interface. If the class is concrete, it must supply all the methods listed in the interface. If the class is abstract, it can leave some of the interface's methods abstract.

CLASS	**INTERFACE**
A superclass provides a secondary data type to objects of its subclasses.	**An interface provides a secondary data type to objects of classes that implement that interface.**
An abstract class cannot be instantiated.	**An interface cannot be instantiated.**
A concrete subclass of an abstract class must define all the inherited abstract methods. (An abstract subclass of an abstract class can leave some of the methods abstract.)	**A concrete class that implements an interface must define all the methods specified by the interface. (An abstract class that implements an interface can leave some of the interface's methods abstract.)**
A class can extend another class. A subclass can add methods and override some of its superclass's methods.	**An interface can extend another interface** (called its *superinterface*) by adding declarations of abstract methods. A *subinterface* cannot override methods of its *superinterface* (because there is nothing to override: all methods are abstract.)
A class can extend only one class.	A class can implement any number of interfaces.
A class can have fields.	An interface cannot have fields except, possibly, some `public static final` constants.
A class defines its own constructors or is supplied with a default no-args constructor by the compiler.	An interface has no constructors.
A concrete class has all its methods defined. An abstract class usually has one or more abstract methods.	All methods declared in an interface are abstract.
Every class is a part of a hierarchy of classes with `Object` at the top.	An interface may belong to a small hierarchy of interfaces, but this is not very common.

The two main similarities are:

1. A concrete class that extends an abstract class and/or implements an interface must supply code for all the abstract methods of its superclass and/or of the interface.

2. Like a superclass, an interface provides a secondary data type to the objects of classes that implement that interface, and polymorphism works for interface types.

Question: Why do we need interfaces? Why not just turn the interface into an abstract class?

Answer: The main difference between superclasses and interfaces is the fact that it is not possible for a class to have more than one superclass, but a class can implement any number of interfaces. Class hierarchies in Java have this limitation: Java assumes that each class neatly falls into one hierarchy. In real life, objects may play different roles in different situations.

Interfaces also help in OOP (Object-Oriented Programming) when a class implements several interfaces, that class must supply the code for all methods of all interfaces implemented. If it happens that the same method is declared in more than one interface, there is not conflict, because the methods of interfaces have no code—only headers.

Interfaces (More information):

- If a concrete class implements several interfaces, it must supply all the methods specified in each of them.
- Polymorphism fully applies to interface data types.