Abstract Classes & Interfaces

- Definitions
  - *abstract methods* = Methods that are declared, with no implementation
  - *abstract class* = A class with abstract methods, not meant to be instantiated
  - *interface* = A named collection of method definitions (without implementations)

<u>Examples</u>

- **Food** is an abstract class. Can you make an instance of food? No, of course not. But you can make an instance of an apple or a steak or a peanut butter cup, which are *types* of food. Food is the abstract concept; it shouldn't exist.
- **Skills** are interfaces. Can you make an instance of a student, an athlete or a chef? No, but you can make an instance of a person, and have that person take on all these skills. Deep down, it's still a person, but this person can also do other things, like study, sprint and cook.

Abstract Classes & Interfaces

- **Q:** So what's the difference between an interface and an abstract class?
- **A:**
  - An interface cannot implement any methods, whereas an abstract class can
  - A class can implement many interfaces but can have only one superclass (abstract or not)
  - An interface is not part of the class hierarchy. Unrelated classes can implement the same interface

- Syntax:
  - abstract class:

    ```
    public class Apple extends Food { … }
    ```
  - interface:

    ```
    public class Person implements
        Student, Athlete, Chef { … }
    ```

Abstract Classes & Interfaces

- **Q:** Why are they useful?
- **A:** By leaving certain methods undefined, these methods can be implemented by several different classes, each in its own way.

<u>Example:</u> Chess Playing Program

– an abstract class called ChessPlayer can have an abstract method `makeMove()`, ***extended*** differently by different subclasses.

```
public abstract class ChessPlayer  {
  <variable declarations>
  <method declarations>
  public void makeMove();        }
```

– an interface called ChessInterface can have a method called `makeMove()`, ***implemented*** differently by different classes.

```
public interface ChessInterface  {
  public void makeMove();        }
```

Abstract Classes & Interfaces

- **Q:** What about inheritance?
- **A:** Follow these simple rules:
  - An abstract class can't inherit from more than one other class.
  - Interfaces can inherit from other interfaces, and a single interface can inherit from multiple other interfaces

<u>Example:</u>

```
interface Singer  {

    void sing();
    void warmUpVoice();
}

interface Dancer  {

    void dance();
    void stretchLegs();
}

interface Talented extends Singer, Dancer  {
    // can sing and dance. Wowwee.
}
```

Abstract Classes & Interfaces

- **Q:** Where else can interfaces be used?
- **A:** You can pass an interface as a parameter or assign a class to an interface variable, just like you would to an abstract class.

Example:

```
Food myLunch = new Sandwich();
Food mySnack = new Apple();
Student steve = new Person();
  //assuming that Person implements Student
```

– If `Person` has methods `eat(Food f)` and `teach(Student s)`, the following is possible:

```
Person bob = new Person();
steve.teach(bob);
steve.eat(myLunch);
System.out.println("Yum.");
```

Data Factories

- **Q:** I'm confused. Why are we using this?
- **A:**

```
Robot r1 = new RobotSE();
Robot r2 = new RobotSE();
Robot r3 = new RobotSE();
…
Robot r100 = new RobotSE();
```

```
Robot r1 =
   DataFactory.makeRobot();
Robot r2 =
   DataFactory.makeRobot();
Robot r3 =
   DataFactory.makeRobot();
…
Robot r100 =
   DataFactory.makeRobot();

public class DataFactory  {
   public RobotSE makeRobot()
     { return new RobotSE(); }
}
```

- Which implementation requires the most line changes if you made a design decision to use a new kind of `Robot` called `SuperRobot` for `r1-r100`?