Announcements

- Program 3 is due 10/12 at 11:59PM
- SDSU STEM Career Fair next week: Wed 16, 2019 10AM - 2:30PM at Montezuma Hall in the Student Union
  - Something useful to know: Github
    - Github is a public remote repository for your coding projects (or really any project)
  - Something useful to know: LinkedIn
    - This a social networking site focused on business networking, I highly recommend you make an account…
  - Include links to both your Github and LinkedIn accounts on your resume

Quiz Time ;)

Polymorphism (revisited)

- **Runtime polymorphism** is where the compiler cannot make the determination but instead the determination is a made while the program is running.

Polymorphism (revisited)

Looking at exceptions we can see another example of runtime polymorphism.

Let’s look at an example...

Interfaces

- An **interface** is similar to an abstract class in that it allows programmers to state that a class adheres to a set of predefined rules.
- An **interface** can specify a set of abstract methods that an implementing class must override and define.
    - In an interface, an abstract method does not need the abstract keyword in front f the method signature
- To create an interface, a programmer uses the keyword "**interface**" in the class definition. For example:
    - public interface Foo { …
    - Notice the lack of the "abstract" and "class" keywords

Interfaces (cont.)

- Any class that implements an interface must:
  - List the interface name after the keyword implements
  - Override and implement the interface's abstract methods

Let's look at an example...

Abstract Classes vs Interfaces

- An abstract class can have implemented code in it, whereas an interface cannot.
- A class can only inherit from ***ONE*** superclass, Java does **not** support multiple inheritance. On the other hand a class can implement multiple interfaces.
    - You can also inherit from a superclass, AND implement multiple interfaces. For example:
    - public class Foo extends Bar implements Interface1, Interface2, Interface3 … etc

Let's modify the previous example...

Abstract Classes and Interfaces (cont.)

Let’s look at Java’s documentation for an ArrayList, we can see the inheritance tree and implemented interfaces

Generics

- A generic type is a generic class or interface that is parameterized over types.
  - In other words, you use a generic type when you do not know what type you will be working with.
- A generic class is any class that uses a generic type.
- The syntax for using a generic type is: <E> applied next to the class name. For example:
  - public class Foo<E> { …
  - The above states that class Foo will be using a generic type called E

Generics (cont.)

The choice of the letter: E as a generic type is based on the following naming convention:

Generic type parameter names should be single, uppercase letters.

- E - element
- K - key
- N - number
- T - type
- V - value
- S, U . etc. - 2nd, and 3rd generic types