Announcements

- Program 3 is due 10/12 at 11:59PM
- Weeks 7-11 Section is now unlocked
- Midterm will be graded in 1 to 2 weeks
- Program 4 will likely be based on Program 3, so make sure you do program 3
- SDSU STEM Career Fair next week: Wed 16, 2019 10AM - 2:30PM at Montezuma Hall in the Student Union

Program 3

- Questions?
- Be sure to write good code, and have good comments / documentation, part of your grade will be me hand grading your code...

Inheritance

Coming back to inheritance to look at: Polymorphism

- **Polymorphism** refers to determining which program behavior to execute depending on data types.
- Method overloading is a form of **compile-time polymorphism**
- **Runtime polymorphism** is where the compiler cannot make the determination but instead the determination is a made while the program is running.

Inheritance (cont.)

Remember, runtime refers to your program while it is running. Compile time is before your program runs, more specifically when it gets compiled into bytecode.

Inheritance (cont.)

- A scenario that requires runtime polymorphism involves inheritance.
- This scenario is a Java feature called **derived/base class reference conversion**, wherein a reference to a derived class can be converted to a reference to the base class (without explicitly casting)
- This is different from other data type conversions, such as converting a double to an int, wherein you need to explicitly cast to the desired type.

Let’s look at an example...

Inheritance (cont.)

Be careful using polymorphism, you must remember that inheritance only goes one way. For example:

- If Undergrad **extends** Student, then the Undergrad class is both: type Undergrad and type Student.
- Where as the Student class is only of type Student, NOT of type undergrad

Let’s look at an example...

Inheritance

The concept of inheritance is commonly confused with the idea of composition.

- Inheritance is the idea that one object is the same type as another object
  - The ‘is-a’ relationship is used to describe inheritance. For example:
  - Foo extends Bar   -   A Foo object ‘is-a’ type of Bar object (But not the other way around right?)
- Composition is the idea that one object has (contains) other objects.
  - The ‘has-a’ relationship is used to describe composition.

Let’s look at an example...

OOP (Object Oriented Programming)

Three big concepts are:

- Inheritance
  - Allows one class to inherit properties and behavior from another class
- Encapsulation
  - A class encapsulates data and behavior to create objects
- Polymorphism
  - Determining different program behavior based on data types

These are three concepts fairly unique to the OOP paradigm, meaning non-OOP languages don’t have these...

Abstract Classes

- An **abstract class** is a class that guides the design of subclasses but cannot itself be instantiated as an object.
- It exists as a superclass that provides a blueprint for creating objects of the same type
    - In other words, while it cannot be instantiated, it does define how subclasses that extend it must be implemented
- A **concrete class** is a class that is not abstract, and therefore can indeed be instantiated
    - This is what we have been using in our class

Abstract Classes (cont.)

- An abstract class is denoted by the keyword **abstract** in front of the class definition.
  - For example: public abstract class Shape{ …
- An **abstract method** exists in an abstract class and is a method that each subclass must implement to be a concrete class.
  - An abstract method is also denoted by the keyword: abstract:
    - abstract double computeArea() {..
- If the a class does not implement the abstract method, then it to must be abstract

Let’s look at an example...

Abstract Classes

Abstract classes are useful for providing runtime polymorphism.

- This allows a programmer to use an abstract method without worrying about which concrete class implements the abstract method

Let’s modify our previous example.