Inheritance

The Big Picture

- object-oriented programming is meant to reflect the structure of things in the real world
  - objects correspond to individual things
  - classes correspond to kinds of things
- in the real world, different kinds of things are not always completely unrelated
  - e.g. apples and fruit – apples are a kind of fruit, though there is fruit that's not apples
  - e.g. savings accounts and checking accounts are both kinds of bank accounts (and there may be other kinds of bank accounts)
- *inheritance* is the mechanism by which we can express "is-a" relationships between classes
- *polymorphism* is the mechanism by which we can write code that works with things related by an "is-a" relationship

Which of the following is the most accurate analogy?

A subclass is to a superclass as

- "apple" is to the apple sitting on my desk
- "fruit" is to the apple sitting on my desk
- the apple sitting on my desk is to "apple"
- the apple sitting on my desk is to "fruit"
- ★ "apple" is to "fruit"
- "fruit" is to "apple"
- "apple" is to "orange"
- "vegetable" is to "fruit"

Inheritance

- inheritance defines an "is-a" relationship between classes

```
public class Apple extends Fruit {
  …
}
```

  - an apple is a (kind of) fruit
- subclasses inherit everything – instance variables and methods – *except* constructors
  - even `private` things, though they cannot be accessed directly
  - new access modifier: `protected` allows only the class and its subclasses to access

Inheritance

Subclasses –

- *can* add new elements (instance variables and methods)
  - a new method has a different header (name and/or number/type of parameters)
- *can* redefine (override) or extend methods
  - same header, new body
  - to extend, also invoke superclass version
- *must* define one or more constructors (in most cases)
  - constructor should first call superclass constructor, then initialize only the instance variables for its own class
- *cannot* redefine instance variables
- *cannot* remove instance variables or methods already defined

When defining a subclass, you can: (choose all that apply)

- ★ add instance variables that are not part of the superclass
- ★ add methods that are not part of the superclass
- ☐ redefine superclass instance variables so they have a different type or purpose in the subclass
- ★ redefine superclass methods so they behave differently in the subclass
- ☐ remove instance variables that are part of the superclass
- ☐ remove methods that are part of the superclass

Inheritance

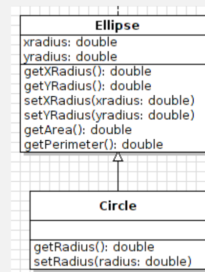
Inheritance is often talked about as a way to reuse existing classes or code – but while this often occurs, it is not why inheritance should be used.

Subclass (only) when **both** –

- "is a", "is a kind of" language makes logical sense, **and**
- everything inherited from the superclass makes sense for the subclass

- Liskov Substitution Principle
- introduced by Barbara Liskov in 1987
- won 2008 Turing Award for work leading to the development of object-oriented programming

Inheritance and the Liskov Substitution Principle



Should Circle extend Ellipse?

Circle "is a kind of" Ellipse...

But Circle inherits setXRadius() and setYRadius(), allowing the following –

```
Circle c = new Circle();
c.setXRadius(5);
c.setYRadius(10);
```

This doesn't make sense for Circle!
(so no, Circle should not extend Ellipse)

Inheritance

- an object is like an onion, with each class in the inheritance hierarchy describing a layer
- top-level class is at the core

- `this` refers to the current layer of the onion
- `super` refers to the next layer in