# Introduction to Inheritance

CST 365 – Web Applications
Michael Ruth, Ph.D.
Assistant Professor
Computer Science & I.T.
mruth@roosevelt.edu

## Objectives

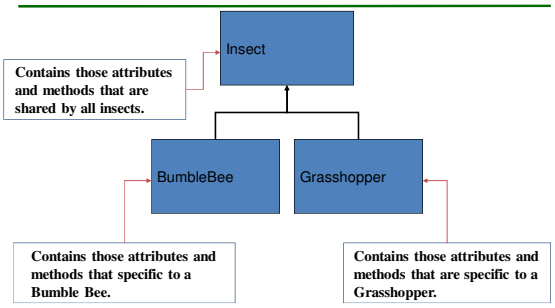
- Discuss the concept of inheritance and the "is-a" relationship in terms of purpose
- Explain the terminology related to the chains of inheritance as it relates to Java
- Discuss class hierarchies and explain their depiction of inheritance relationships
- Explain the role that class Object plays in inheritance hierarchies and its importance
- Discuss the OOP concept of polymorphism as it applies to all forms of inheritance



## What is Inheritance?

- ***Generalization vs. Specialization***
- Real-life objects are typically specialized versions of other more general objects.
- The term "insect" describes a very general type of creature with numerous characteristics.
- Grasshoppers and bumblebees are insects
  - They share the general characteristics of an insect.
  - However, they have special characteristics of their own.
    - grasshoppers have a jumping ability, and
    - bumblebees have a stinger.
- Grasshoppers and bumblebees are specialized versions of an insect

# Inheritance






# The "is a" Relationship

- The relationship between a superclass and an inherited class is called an ***"is a" relationship***
  - A grasshopper "is a" insect.
  - A poodle "is a" dog.
  - A car "is a" vehicle.
- A specialized object has:
  - all of the characteristics of the general object, plus
  - additional characteristics that make it special
- In object-oriented programming, ***inheritance*** is used to create an "is a" relationship among classes



# The "is a" Relationship

- We can ***extend*** the capabilities of a class.
- Inheritance involves a superclass and a subclass.
  - The ***superclass*** is the general class and
  - the ***subclass*** is the specialized class
- The ***subclass*** extends the ***superclass***
  - ***superclasses*** are also called ***base*** classes
  - ***subclasses*** are also called ***derived*** classes
- The relationship of classes can be thought of as ***parent*** classes and ***child*** classes

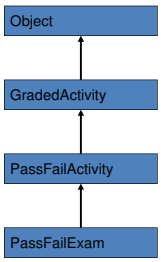
## Inheritance

- The subclass ***inherits*** fields and methods from the superclass
  - New fields and methods may be added to the subclass



## Chains of Inheritance

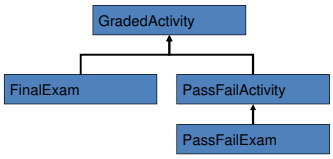
- A superclass can also be derived from another class






## Chains of Inheritance

- Classes often are depicted graphically in a ***class hierarchy***
  - Shows the inheritance relationships between classes



## The `Object` Class

- All Java classes are directly or indirectly derived from a class named `Object`
  - Object is in the `java.lang` package.
- Ultimately, every class is derived from the `Object` class
  - Classes without extends derive directly



## The `Object` Class

- Because every class is directly or indirectly derived from the Object class:
  - every class inherits the Object class's members
    - example: toString and equals
- In the Object class, the ***toString*** method returns a string containing the object's class name and a hash of its memory address
- The ***equals*** method accepts the address of an object as its argument and returns true if it is the same as the calling object's address



## Abstract Methods/Classes

- Since there will be no implementation in the superclass, a dummy method is required
  - This dummy method is said to be **abstract**
- Abstract methods:
  - Force subclasses to override the method
  - Have no implementation/body
  - Uses the keyword **abstract**
- Abstract Class
  - Any class that has at least one abstract method is an abstract class
  - These classes cannot be instantiated! They're abstract!
  - Subclasses MUST override ALL abstract methods
  - Uses the keyword **abstract**

## Programming with Interfaces

- In the design phase, we try to develop interfaces as part of the design
  - Programming to interfaces allows developers to work on separate parts independently
  - Interfaces should be a stable part of design!
- This allows the independent developers to remain independent!



## Interfaces (Cont.)

- An interface looks similar to a class, except:
  - keyword **interface** is used instead of the keyword **class**
  - the methods that are specified have no bodies, only signatures terminated with semicolons
    - Note all methods are public by default…
- The general format of an interface definition:

```
public interface InterfaceName
{
   (Method headers...)
}
```



## Polymorphism

- The term ***polymorphism*** means the ability to take many forms
- In Java, reference variables can reference objects of classes that are derived from the variable's class
  - Variables called ***polymorphic*** variables

## Summary

- Discussed the concept of inheritance and the "is-a" relationship in terms of purpose
- Explained the terminology related to the chains of inheritance as it relates to Java
- Discussed class hierarchies and explain their depiction of inheritance relationships
- Explained the role that class Object plays in inheritance hierarchies and its importance
- Discussed the OOP concept of polymorphism as it applies to all forms of inheritance



## Questions?