Inheritance

Inheritance allows you to write new classes based on existing (super or base) classes
- Inherit super class methods and data
- Add new methods and data

This allows substantial reuse of Java code
- When extending software, we often write new code that invokes old code (libraries, etc.)
- We sometimes need to have old code invoke new code (even code that wasn’t imagined when the old code was written), without changing (or even having) the old code
  - E.g., A drawing program must manage a new shape
  - Inheritance allows us to do this also
Access for inheritance

- Class may contain members (methods or data) of type:
  - Private:
    - Access only by class’s methods
  - Protected
    - Access by:
      - Class’s methods
      - Methods of inheriting classes, called subclasses or derived classes
      - Classes in same package
  - Package:
    - Access by methods of classes in same package
  - Public:
    - Access to all classes everywhere

A Programming Project

- Department has system with Student class
  - Has extensive data (name, ID, courses, year, …) for all students that you need to use/display
  - Department wants to manage research projects better
    - Undergrads and grads have very different roles
      - Positions, credit/grading, pay, …
  - You want to reuse the Student class but need to add very different data and methods by grad/undergrad
    - Suppose Student was written 5 years ago by someone else without any knowledge that it might be used to manage research projects
public ... main(...){
    Student S1 = new Student ("Jo", "Wang", 1);
    ...
    S1.printData();
    // Prints all data
    ...
}

Classes and Objects

Encapsulation  Message passing  Main method

Student S1
private:
last Name
firstName
dept
public:
printData

public ...

Classes and Objects

Inheritance

Already written:
Class Student
  firstName
  lastName
  dept
  printData
  ...

Class Undergrad
  firstName
  lastName
  dept
  underWage
  underHours
  printData
  getPay

Class Grad
  firstName
  lastName
  dept
  gradSalary
  printData
  getPay

You next write:
Inheritance, p.2

Exercise: Student class

- Write a public `Student` class as a base or super class:
  - Two private variables: first name, last name
  - Constructor with two arguments
  - Void method `printData()` to print the first + last name:
Exercise: Undergrad class

• Write an Undergrad class as a derived or subclass:
  – Class declaration:
    • public class Undergrad extends Student
  – Add private double variables underWage and underHours
  – Constructor: How many arguments does it have?
    • Invoke superclass constructor in 1st line of body:
      super( <arguments> ) // Use actual arguments
    • And then set the two new private variables as usual
  – Method getPay() returns double underWage * underHours
  – Method printData() prints name and pay (void)
    • Use superclass printData() method to print name in 1st line:
      super.printData();
    • Write a second line to System.out.println weekly pay

Exercise: Grad class

• Write a Grad class as a derived or subclass:
  – Class declaration: extends Student
  – Add private double variable gradSalary
  – Constructor: How many arguments does it have?
    • Invoke superclass constructor in 1st line of body:
      super( <arguments> ) // Use actual args
    • And then set the new private variable
  – Method getPay() returns double gradSalary
  – Method printData() prints name and pay (void)
    • Use superclass printData() method to print name on 1st line
    • Write second line to print monthly pay
Exercise: Special Grad class

**Write SpecGrad class as derived or subclass:**
- **Class declaration:** extends ______
- **Add private double variable** specStipend
- **Constructor:** *How many arguments does it have?*
  - Invoke superclass constructor: super(<arguments>)
  - And then set the new private variable
- **Method** getPay() returns double specStipend
- **Method** printData() prints name and pay (void)
  - Use superclass printData() method to print name and monthly salary (which is zero)
  - Write second line to print stipend
- A special grad gets only a stipend, not a monthly salary. We’ll discuss it in solutions.

Exercise: main()

**Download class StudentTest**
- It has only a main() method, which:
  - Creates Undergrad ferd at $12/hr for 8 hrs
  - Prints Ferd’s data
  - Creates Grad ann at $1500/month
  - Prints Ann’s data
  - Creates SpecGrad mary at $2000/term
  - Prints Mary’s data
  - Creates an array of 3 Students
  - Sets array elements to ferd, ann, mary
  - Loops through the array and uses printData() on each Student object in the array to show their data.
- What happens in the loop? Did you expect it?
Main method

```java
public class StudentTest {
    public static void main(String[] args) {
        Undergrad ferd = new Undergrad("Ferd", "Smith", 12.00, 8.0);
        ferd.printData();
        Grad ann = new Grad("Ann", "Brown", 1500.00);
        ann.printData();
        SpecGrad mary = new SpecGrad("Mary", "Barrett", 2000.00);
        mary.printData();
        System.out.println();

        // Polymorphism, and late binding
        Student[] team = new Student[3];
        team[0] = ferd;
        team[1] = ann;
        team[2] = mary;
        for (int i = 0; i < 3; i++)
            team[i].printData();
    }
}
```

Java has internal table with the most specific object type and chooses the appropriate method at run time.

Inheritance: Type set at runtime

- We can write a variation on StudentTest to prompt the user to pick a student type (undergrad, grad, special grad) with a JOptionPane, and then enter the needed data
  - The Undergrad, Grad or SpecGrad object would be placed in the team array
- When this program is compiled it has no way of knowing what kinds of Students will be added to the team array by a user
- When the program is run and objects are added, their types are dynamically tracked
  - In the team array, each object’s specific printData() method will be invoked
StudentTest with input

```java
import javax.swing.*;
public class StudentTestWithInput {
    public static void main(String[] args) {
        Student[] team = new Student[3];
        for (int i = 0; i < team.length; i++) {
            String type = JOptionPane.showInputDialog("Enter type");
            String fname = JOptionPane.showInputDialog("Enter fname");
            Stringlname = JOptionPane.showInputDialog("Enter lname");
            String payStr = JOptionPane.showInputDialog("Enter pay");
            double pay = Double.parseDouble(payStr);
            if (type.equals("Grad"))
                team[i] = new Grad(fname, lname, pay);
            else if (type.equals("SpecGrad"))
                team[i] = new SpecGrad(fname, lname, pay);
            else
                team[i] = new Undergrad(fname, lname, pay, 8.0);
        }
        // Polymorphism, and late binding
        for (int i = 0; i < 3; i++) {
            System.out.print(team[i].getClass() + ":
            ");
            team[i].printData();
        }
    }
}
```

Exercise

- In class Grad:
  - Change printData() to use getPay() instead of explicitly printing gradSalary
  - Save/compile and run StudentTest
  - What happens?
  - Why?