1.00 Lecture 13

Inheritance

Reading for next time: Big Java: sections 11.5-11.6

Inheritance

- **Inheritance allows you to write new classes based on existing (super) 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!**

Access for inheritance

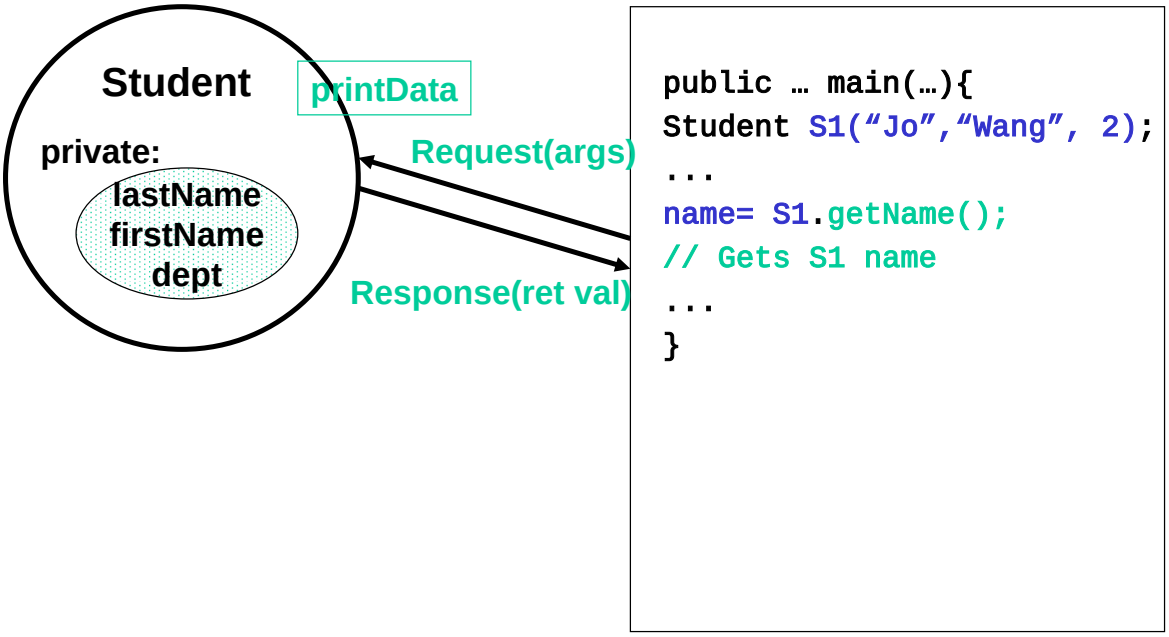
- **Class may contain members (methods or data) of type:**
  - **Private:**
    - **Access only by class's methods**
  - **Protected (rarely used in Java; it's pretty unsafe)**
    - **Access by:**
      - **Class's methods**
      - **Methods of inherited classes, called subclasses**
      - **Classes in same package [this is a problem in my view]**
  - **Package:**
    - **Access by methods of classes in same package**
  - **Public:**
    - **Access to all classes everywhere**

A Programming Project

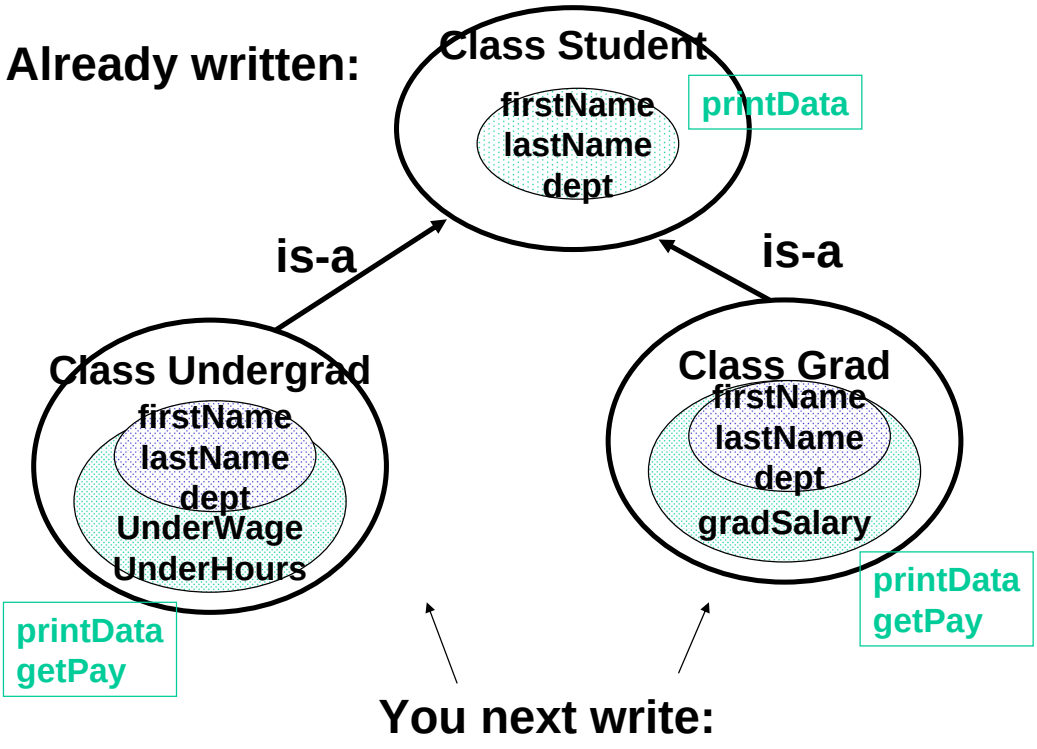
- **Department has system with Student class**
  - **Has extensive data (name, ID, courses, year, …) for all students that you need to use/display**
  - **Dept wants to manage research projects better**
    - **Undergrads and grads have very different roles**
      - **Positions, credit/grading, pay, …**
  - **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**

Classes and Objects

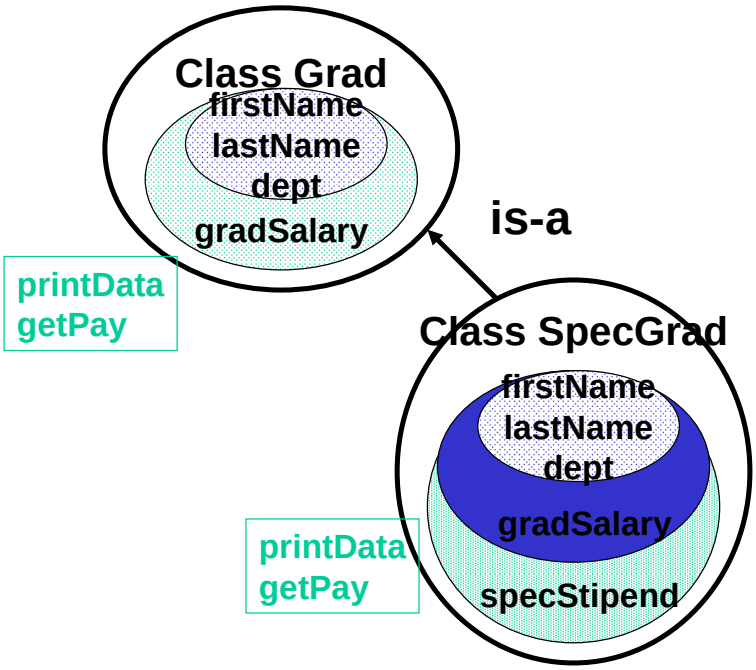
Encapsulation Message passing "Main event loop"



Inheritance



Inheritance, p.2



Exercise: Student class

- **Write a `Student` class as a base 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 class:**
  - **Class declaration `extends Student`**
  - **Add private variables `underWage` and `underHours`**
  - **Constructor: *How many arguments does it have?***
    - **Invokes superclass constructor: `super(arguments)`**
    - **Sets the two new private variables**
  - **Method `getPay()` returns `double underWage * underHours`**
  - **Method `printData()` prints name and `pay` (void)**
    - **Use superclass printData() method to print name:**

```
super.printData();
```

Exercise: Grad class

- **Write a `Grad` class as a derived class:**
  - **Class declaration '`extends Student`'**
  - **Add private variable `gradSalary`**
  - **Constructor: *How many arguments does it have?***
    - **Invokes superclass constructor: `super(arguments)`**
    - **Sets the new private variable**
  - **Method `getPay()` returns `double gradSalary`**
  - **Method `printData()` prints name and `pay` (void)**
    - **Use superclass `printData()` method to print name**

Exercise: Special Grad class

- **Write a `SpecialGrad` class as a derived class of `Grad`:**
  - **Class declaration '`extends _____`'**
  - **Add private variable `specStipend`**
  - **Constructor: *How many arguments does it have?***
    - **Invokes superclass constructor: `super(arguments)`**
    - **Sets the new private variable**
  - **Method `getPay()` returns `double specStipend`**
  - **Method `printData()` prints name and `pay` (void)**
    - **Use superclass `printData()` method to print name**

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 `SpecialGrad 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?**

Solution: Main method

```
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, or 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 knows the object type and chooses the appropriate method at run time

Output from main method

```
Ferd Smith
Weekly pay: $96.0
Ann Brown
Monthly salary: $1500.0
Mary Barrett
Monthly salary: $0.0
Semester stipend: $2000.0
```

Note that we could not write:

```
team[i].getPay();
```

because getPay() is not a method of the superclass Student. In contrast, printData() is a method of Student, so Java can find the appropriate version.

We'd have similar problems with a method like isUROP that would only be defined for undergrads and not in Student

Optional exercise

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