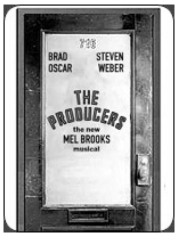
Theatre Payroll App

Before we begin

- Questions?
- LDAP database…problems?
- Attendance

Something to think about for next class!

- Any theatre fans in the house?



When we last left our wannabe producer

- "I Want to be a Producer"
  - Business manager for a theatrical production.
  - Need a simple application that allows me to determine the total amount spent on performers salaries for a given week.
  - Performers are paid a flat rate per performance given.

When we last left our wannabe producer

- I Want to be a Producer"
  - The app will need to:
    - Be able to accept the number of performances given by each performer
    - Calculate the salary paid for each performer
    - Determine the total salary paid for all performers
  - Any thoughts on classes we might wish to define to implement this?

Theatre Payroll app Classes

- Actor
  - Represents an actor that needs to be paid
- Payroll
  - Object that manages all actors that need to be paid
- Javadocs for these classes are in the handout

Classes

- Recall that for classes, one can define
  - Instance variables
  - Class variables
  - Methods
- Looking at the Javadocs, what are the instance variables, class variables, and methods for the Actor and Payroll classes?

The Actor Class

- Member variables
  - Name
  - nPerformances
- Class Variables
  - PAYRATE
- Methods
  - perform(int n) – sets number of performances
  - calculatePay()

The Payroll Class

- Member variables
  - performers – Array of actors managed by payroll
  - nPerf – Number of actors being managed
  - MAXPERF – The maximum number of actors the payroll can handle.
- Methods
  - addPerformer (Peformer P) – add a performer to the payroll.
  - calculateTotalPay()
  - main(String args[]) – used to test

A look at the code

- First, consider style

```
/*
 * Actor.java
 *
 * Version:
 *     $Id: $
 *
 * Revisions:
 *     $Log: $
 */
```

RCS

- These comments are here for RCS
- Revision Control System
- A means for managing document versioning
- Check-in / Check-out
- Why this is a good idea
  - Software versioning
  - Management of group software projects

A further look at the code

- More style

```
/**
 * Simple Actor class.  Represents an actor
 * that needs to get paid
 *
 * @author Joe Geigel
 */
public class Actor {…

/**
  *
  * Constructor for Actor class
  *
  * @param name The actor's name
  */
```

Javadoc

- More than just good style, these comments are used to create Javadocs
  - If you comment your code well using these conventions, you have instant documentation!
  - Compare comments in code with attached Javadocs.

Finally, a look at the code

- Actor – instance / class variables

```
/**
 * The basic rate per perfomance.
 * Common for all actors
 */
private static final double PAYRATE = 200.0;

/**
 * The name of the actor
 */
private String myName;

/**
 * The number of perfomances worked
 */
private int nPerformances;
```

More code

- Actor – constructor

```
/**
 *
 * Constructor for performer class
 *
 * @param name The actor's name
 */
public Actor (String name){
   myName = name;
   nPerformances = 0;
}
```

More code

- Actor – defining number of performances

```
/**
 * Sets the number of performances for the week
 *
 * @param n number of performances played during
 * the week
 */
public void perform (int n)
{
   nPerformance = n;
}
```

Even More code

- Actor – calculating the weekly pay

```
/**
 * Calculates and returns the weekly pay for the
 * actor
 *
 * @returns The weekly pay
 */
public double calculatePay ()
{
   return PAYRATE * nPerformances;
}
```

Now..to the payroll

- Payroll – instance / class variables

```
/**
 * An array contain the managed actors
 */
private Actor actors[];

/**
 * The number of actors currently being managed
 */
private int nActors;

/**
 * The maximum allowable actors that can be managed
 */
private static final int MAXACTOR = 100;
```

More Payroll code

- Payroll -- constructor

```
/**
 *
 * Default constructor for the Payroll class
 *
 */
public Payroll () {
   actors = new Actor[MAXACTOR];
   nActors = 0;
}
```

More payroll code

- Payroll – adding a Performer

```
  /**
   * Adds an actor to the payroll.  Will issue an
   * error message if the payroll is currently full.
   *
   * @param A the actor to be added
   */
public void addActor (Actor A)
{
  if (nActors == MAXACTOR)
    System.err.println ("Payroll is full.”);
  else {
     actors[nActors] = A;
     nActors++;
  }
}
```

The last of the payroll code

- Payroll – calculating total pay

```
/**
 *  Caculates the weekly pay for all of the actors
 *
 * @returns the total weekly pay
 */
public double calculateTotalPay()
{
   double sum = 0.0;
   for (int i=0; i < nActors; i++)
        sum += actors[i].calculatePay();
   return sum;
}
```

Testing the code

- Using the Payroll.main method

```
  /**
   * A test program for the Payroll Class
   */
static public void main (String args[])
{
   // Create a payroll
   Payroll pay = new Payroll();

   // Create some performers, define the number of
   // performances for each then add them to the payroll
   Actor A = new Actor("Nathan Lane");
   A.perform (8);
   pay.addActor (A);
   ...

   // Calculate and print out the total weekly pay
   System.out.println ("The total weekly pay for this week is " +
                pay.calculateTotalPay());
 }
```

Testing the code

- To run the test, issue the command:
  - `java Payroll`

  - `The total weekly pay for this week is 4600.0`

- Questions?

Adding to the Producer app

- The real producers liked the app so much that they want to add musicians to the mix:
  – However:
    - Musicians get reimbursed for instrument rental as well as getting a base pay per performance
    - The pay rate for musicians is different than that for actors.

Potential Problems

- Musicians
  - Have data items that actors do not
  - Determine their pay differently than actors
- Yet…
  - The producer still needs to maintain pay info about musician AND actors

Consider this code in payroll

```
/**
 *  Caculates the weekly pay for all of the actors
 *
 * @returns the total weekly pay
 */
public double calculateTotalPay()
{
   double sum = 0.0;
   for (int i=0; i < nActors; i++)
        sum += actors[i].calculatePay();
   return sum;
}
```

Adding musicians does not change this algorithm!

What would be nice

- Keep the same code
- Let each object in the array compute it's pay based on the kind of object it is.
- We can achieve this using inheritance / subclassing.

Something to think about for next class!

- We enhance the app while maximizing use of the code already written by using
- Inheritance and Polymorphism…
  - But that's for next time.
- Questions?