Inheritance and Polymorphism in Java

IFT 194: Lab 5

Brandon Doyle
bdoyle5@asu.edu
1215232174

Dr. Usha Jagannathan
Usha.Jagannathan@asu.edu

July 29, 2018

Summary

View the source of this document on GitHub.

Overriding the *equals* method

In this question we're asked to write a class `Player` that holds information about an athlete. The class must have an `equals`-method that compares two instances of `Player`. Please see Figure 2 for my solution. The following is an example session.

```
Please enter a player name: Brandon
Please enter a team name: ASU
Please enter a jersey number: 55

Please enter a player name: Jasmine
Please enter a team name: ASU
Please enter a jersey number: 27

Name:      Brandon
Team Name: ASU
Jersey:    55

Name:      Jasmine
Team Name: ASU
Jersey:    27

Brandon == Jasmine: false
```

Figure 1: Example output from `Player.java` in Figure 2.

Although this code seemingly works well, in practice I believe duplication may be prevented using metaclasses, much in the same way metaclasses may be used to create singletons.

This part of the lab is straightforward since we've already previously considered the difference between, for example, `"example1" == "example2"` and `"example1".equals("example2")`.

Also, I would again like to highlight the rather useful differences between creating `String` objects in Java with `new String("ex")` as opposed to the more brief `"ex"` (i.e., not encapsulating the instantation with an explicit call to `String`): the former always creates a new instance, whereas the latter makes an implicit call to *intern* the sequence. In other words, if memory or space isn't so much of an issue, yet performance is, I might use the former instantiation over the latter to prevent interning, and vice-versa.

Another type of employee

In this question we're tasked with extending the class hierarchy included in `Staff.java` in Figure 4. (I condensed the classes to a single file so I could see the hierarchy/outline.) See Figure 3 for an example.

Conclusion

It took about 4 hours to complete this lab. One thing I learned more about is `abstract` classes. I've known of this feature in Python (cf. abc), but I've never used the same concept in Java before. For example, see Figure 5 for a rewrite of `StaffMember` in Python.

I didn't really come across any issues with these concepts, i.e. inheritance and polymorphism, but that's probably because I've seen them in a few different languages.

```
Name:    Clint
Address: 1 Lomb Mem. Dr.
Phone:   585-5903
Current hours: 14
Total Sales: 456.62
$2,914.02

Name:    Marylin
Address: Mars
Phone:   inf
Current hours: 40
Total Sales: 950.0
$5,332.40
```

Figure 3: Example output from `Staff.java` in Figure 4.

```java
package lab_5;

import java.util.InputMismatchException;
import java.util.Scanner;

public class Player
{
    private String _name = "";
    private String _team = "";
    private int _jerseyNumber = 0;

    private Scanner _scnr;

    /**
     * Set up a Scanner for the Player type.
     */
    public Player(Scanner scnr)
    {
        this._scnr = scnr;
    }

    /**
     * Prompt-for and read the player’s data.
     */
    public void readPlayer()
    {
        this._name = getName("player name");
        this._team = getName("team name");
        this._jerseyNumber = getNumber("jersey number");
    }

    /**
     * Read a valid positive integer from the user.
     *
     * @param prompt A prompt.
     * @return A valid, positive integer.
     */
    private int getNumber(String prompt)
    {
        int out;

        do {
            System.out.print("Please enter a " + prompt + ": ");
            try {
                out = this._scnr.nextInt();
                if (out > 0) {
                    break;
                } else {
                    System.out.println(
                        "*** Error: Please enter a positive integer"
                    );
                }
            } catch (InputMismatchException ex) {
                System.out.println("*** Error: Please enter a valid integer");
                this._scnr.next();
            }
        } while (true);

        return out;
    }

    /**
     * Read in a name.
     *
     * @param  A prompt.
     * @return The name.
     */
    private String getName(String prompt)
    {
        String out;

        do {
            System.out.print("Please enter a " + prompt + ": ");
            out = this._scnr.next();
            if (out.length() != 0)
                break;
        } while (true);

        return out;
    }
    public String getName()
    {
        return this._name;
    }

    public String getTeamName()
    {
        return this._team;
    }

    public int getJerseyNumber()
    {
        return this._jerseyNumber;
    }

    @Override
    public String toString()
    {
        return "Name:      " + this._name + "\n"
             + "Team Name: " + this._team + "\n"
             + "Jersey:    " + this._jerseyNumber;
    }

    /**
     * If contents of this player are the same as another, then they represent the
     * same player.
     *
     * @param other Another Player.
     * @return Whether the players are the same player.
     */
    public boolean equals(Player other)
    {
        return this._jerseyNumber == other.getJerseyNumber()
            && this._team.equals(other.getTeamName());
    }

    /**
     * Provide the same functionality as ComparePlayers.java.
     *
     * @param args Not used.
     */
    public static void main(String[] args)
    {
        try (var scnr = new Scanner(System.in)) {
            var player1 = new Player(scnr);
            player1.readPlayer();
            System.out.println();
            var player2 = new Player(scnr);
            player2.readPlayer();

            System.out.println();
            System.out.println(player1);
            System.out.println();
            System.out.println(player2);
            System.out.println();
            System.out.printf("%s == %s: %b", player1.getName(), player2.getName(),
                    player1.equals(player2));
        }
    }
}
```

Figure 2: Player.java

```java
package lab_5;

import java.text.DecimalFormat;

public class Staff
{
    final private static int NUM_STAFF = 8;

    StaffMember[] staffList;

    public Staff()
    {
        staffList = new StaffMember[Staff.NUM_STAFF];

        // Initialize our staff
        staffList[0] = new Executive("Sam", "123 Main Line", "555-0469",
                "123-45-6789", 2423.07);

        staffList[1] = new Employee("Carla", "456 Off Line", "555-0101",
                "987-65-4321", 1246.15);

        staffList[2] = new Employee("Woody", "789 Off Rocker", "555-0000",
                "010-20-3040", 1169.23);

        staffList[3] = new Hourly("Diane", "678 Fifth Ave.", "555-0690",
                "958-47-3625", 10.55);

        staffList[4] = new Volunteer("Norm", "987 Suds Blvd", "555-8374");

        staffList[5] = new Volunteer("Cliff", "321 Duds Lane", "555-7282");

        staffList[6] = new Commission("Clint", "1 Lomb Mem. Dr.", "585-5903",
                "123-45-6780", 12.45, 6.0);

        staffList[7] = new Commission("Marylin", "Mars", "inf", "123-45-6781",
                14.56, 5.0);

        ((Commission)staffList[6]).addHours(14);
        ((Commission)staffList[6]).addSales(456.62);

        ((Commission)staffList[7]).addHours(40);
        ((Commission)staffList[7]).addSales(950.00);

        System.out.println(staffList[6]);
        System.out.println(DecimalFormat.getCurrencyInstance().format(((Commission)staffList
             [6]).pay()));
        System.out.println();
        System.out.println(staffList[7]);
        System.out.println(DecimalFormat.getCurrencyInstance().format(((Commission)staffList
             [7]).pay()));
    }

    public static void main(String[] args)
    {
        var inst = new Staff();
    }
}


/**
 * An abstract class to form a basis for staff members in our company.
 *
 * @author Brandon Doyle
 */
abstract class StaffMember
{
    protected String name;
    protected String address;
    protected String phone;

    public StaffMember(String eName, String eAddress, String ePhone)
    {
        this.name = eName;
        this.address = eAddress;
        this.phone = ePhone;
    }

    @Override
    public String toString()
    {
        return "Name:    " + this.name    + "\n"
```

```java
             + "Address: " + this.address + "\n"
             + "Phone:   " + this.phone;
    }

    public abstract double pay();
}


/**
 * A general employee.
 *
 * @author Brandon Doyle
 */
class Employee extends StaffMember
{
    protected String socialSecurityNumber;
    protected double payRate;

    /**
     * Override the parent abstract class' constructor.
     *
     * @param eName        Name of the employee.
     * @param eAddress     Address of the employee.
     * @param ePhone       Phone number of the employee.
     * @param socSecNumber Social Security Number of the employee.
     * @param rate         Rate of pay for the employee.
     */
    public Employee(String eName, String eAddress, String ePhone,
            String socSecNumber, double rate)
    {
        super(eName, eAddress, ePhone);
        this.socialSecurityNumber = socSecNumber;
        this.payRate = rate;
    }

    @Override
    public double pay()
    {
        return this.payRate;
    }
}


/**
 * An hourly employee.
 *
 * @author Brandon Doyle
 */
class Hourly extends Employee
{
    protected int _hoursWorked;

    public Hourly(String eName, String eAddress, String ePhone, String socSecNumber,
            double rate)
    {
        super(eName, eAddress, ePhone, socSecNumber, rate);
        this._hoursWorked = 0;
    }

    /**
     * Add hours to our hourly worker's pay period.
     *
     * @param moreHours Number of hours to add.
     */
    public void addHours(int moreHours)
    {
        this._hoursWorked += moreHours;
    }

    /**
     * Compute the pay of an hourly worker.
     */
    @Override
    public double pay()
    {
        double payment = this.payRate * this._hoursWorked;
        this._hoursWorked = 0;
        return payment;
    }

    @Override
    public String toString()
    {
        String result = super.toString();
        result += "\nCurrent hours: " + this._hoursWorked;
        return result;
    }
}


/**
 * Extend our Hourly class of staff members to include commissioned workers.
 *
 * @author Brandon Doyle
 */
class Commission extends Hourly
{
    private double _totalSales;
    private double _commissionRate;

    public Commission(String eName, String eAddress, String ePhone,
            String socSecNumber, double rate, double commission)
    {
        super(eName, eAddress, ePhone, socSecNumber, rate);
        this._commissionRate = commission;
        this._totalSales = 0.0; // wouldn’t an integer be better?
    }

    /**
     * Add sales to this employee’s total.
     *
     * @param totalSales The sales to add.
     */
    public void addSales(double totalSales)
    {
        this._totalSales += totalSales;
    }

    /**
     * Compute the pay of a commissioned worker. Also note that we don’t need to
     * reset _hoursWorked, since it is implicitly reset in the super call to Hourly.
     */
    @Override
    public double pay()
    {
        double total = super.pay() + this._commissionRate * this._totalSales;
        this._totalSales = 0.0;
        return total;
    }

    @Override
    public String toString()
    {
        String result = super.toString();
        result += "\nTotal Sales: " + this._totalSales;
        return result;
    }
}


/**
 * A volunteer in our company.
 *
 * @author Brandon Doyle
 */
class Volunteer extends StaffMember
{
    public Volunteer(String eName, String eAddress, String ePhone)
    {
        super(eName, eAddress, ePhone);
    }

    @Override
    public double pay()
    {
        return 0.0;
    }
}


/**
 * An executive in our company.
 *
 * @author Brandon Doyle
 */
class Executive extends Employee
{
    private double _bonus;

    public Executive(String eName, String eAddress, String ePhone,
            String socSecNumber, double rate)
    {
        super(eName, eAddress, ePhone, socSecNumber, rate);
        this._bonus = 0;    // Has yet to be awarded
    }

    /**
     * Award a bonus to the business executive.
     *
     * @param execBonus Amount of the bonus.
     * @throws IllegalArgumentException If the bonus is less than 0.
     */
    public void awardBonus(double execBonus) throws IllegalArgumentException
    {
        if (execBonus < 0.0)
            throw new IllegalArgumentException("Must provide a positive bonus");
        this._bonus = execBonus;
    }

    /**
     * Override Employee's pay method to support a bonus. Note that if ‘awardBonus‘
     * has not yet been called with an update, the default is 0.
     */
    @Override
    public double pay()
    {
        double payment = super.pay() + this._bonus;
        this._bonus = 0.0;
        return payment;
    }
}
```

Figure 4: Staff.java.

```python
#! /usr/bin/env python3.6
# -*- coding: utf-8 -*

from abc import abstractmethod, ABCMeta


class StaffMember(metaclass=ABCMeta):
    """
    A rewrite of the abstract StaffMember class in Java.
    """
    def __init__(self, eName: str, eAddress: str, ePhone: str) -> None:
        self.name = eName
        self.address = eAddress
        self.phone = ePhone

    def __str__(self) -> str:
        return f'Name: {self.name}\nAddress: {self.address}\nPhone: {self.phone}'

    @abstractmethod
    def pay(self) -> float:
        """
        Compute the pay of this employee.
        """
        raise NotImplementedError


class Volunteer(StaffMember):
    """
    A volunteer in our organization.
    """

    def pay(self) -> float:
        return 0.0


if __name__ == '__main__':
    # Trying to instantiate StaffMember during runtime raises an exception
    #member = StaffMember('Brandon', 'New York', '555-5555')

    # However the following runs without an exception
    member = Volunteer('Brandon', 'New York', '555-5555')
```

Figure 5: staff.py.