OOP Design and Interfaces
IFT 194: Lab 4

Brandon Doyle
bdoyle5@asu.edu
1215232174

Dr. Usha Jagannathan
Usha.Jagannathan@asu.edu

July 27, 2018

Summary

View the source of this document on GitHub.

Changing People

In this section we're asked to draw a trace of the program in `ChangingPeople.java`, which instantiates and modifies instances of `Person`. I've included both of these classes in Figure 2 and Figure 3.

```
person1 -> Person("Sally", 13)
person2 -> Person("Sam", 15)
age := 21
name -> "Jill"

               |  (Method call)
               V

{ person1, p1 } -> Person("Sally", 13)
{ person2, p2 } -> Person("Sam", 15)
age := 21
{ name, name' } -> "Jill"
age' := 21

               |  Changes to p2
               V

{ person1, p1 } -> Person("Sally", 13)
person2 -> Person("Sam", 15)
age := 21
{ name, name' } -> "Jill"
age' := 21
{ p3, p2 } -> Person(name', 21)  ###

               |  Changes to p1
               V

{ person1, p1 } -> Person(name', 55)
person2 -> Person("Sam", 15)
age := 21
name -> "Jill"
name' -> "Jack"
age' := 55
{ p3, p2 } -> Person(name, 21)

               |  Return to main method
               V

person1 -> Person("Jack", 55)
person2 -> Person("Sam", 15)
age := 21
name -> "Jill"
```

Figure 1: Trace of `ChangingPeople.java`. Regarding the line delimited by `###`, I'm assuming that, because this type of String instantiation is implicitly interned, it's the same instance as that referred to by name. The primitive value `21` is copied into the new object. Moreover, as a note on notation: identifiers that lie in braces are references to the same object.

Using the **`Comparable`** Interface

1. In this question we're asked to write a class with a `static` method `largest` that has the ability to return the largest of three arguments, which should be a type that implements the

`Comparable<>` interface. See Figure 4 and Figure 5 for my solution.

A Flexible Account Class

Opening and Closing Accounts

Transferring Funds

```java
package lab_4;

/**
 * Represent a person.
 *
 * @author Brandon Doyle
 */
public class Person
{
    private String _name;
    private int _age;

    /**
     * Class constructor to initialize fields.
     *
     * @param name Name of the person this object shall represent.
     * @param age  Age of the person.
     */
    public Person(String name, int age)
    {
        this._name = name;
        this._age  = age;
    }

    /**
     * Setter to modify the _name field.
     *
     * @param newName New name we'd like this Person to have.
     */
    public void changeName(String newName)
    {
        this._name = newName;
    }

    /**
     * Setter to modify the _age field.
     *
     * @param newAge New age we'd like this Person to have.
     */
    public void changeAge(int newAge)
    {
        this._age = newAge;
    }

    @Override
    public String toString()
    {
        return this._name + " - Age " + this._age;
    }
}
```

Figure 2: Person.java

```java
package lab_4;

public class ChangingPeople
{
    public static void main(String[] args)
    {
        var person1 = new Person("Sally", 13);
        var person2 = new Person("Sam", 15);
        int age = 21;
        var name = "Jill";

        // Original instantiations
        System.out.println("Original values:");
        System.out.println("person1: " + person1);
        System.out.println("person2: " + person2);
        System.out.println("age: " + age + "\tname: " + name + "\n");

        // Modify these values
        ChangingPeople.changePeople(person1, person2, age, name);

        // After modifications
        System.out.println("Values after calling changePeople:");
        System.out.println("person1: " + person1);
        System.out.println("person2: " + person2);
        System.out.println("age: " + age + "\tname: " + name);
    }

    public static void changePeople(Person p1, Person p2, int age, String name)
    {
        // Show the original values
        System.out.println("Original values:");
        System.out.println("person1: " + p1);
        System.out.println("person2: " + p2);
        System.out.println("age: " + age + "\tname: " + name + "\n");

        /* Changes to p2 */
        var p3 = new Person(name, age);
        p2 = p3;    // modify reference(p2) := reference(p3)

        /* Changes to P1 */
        name = "Jack";
        age = 55;
        p1.changeName(name);
        p1.changeAge(age);

        // Print changes
        System.out.println("Inside changePeople:");
        System.out.println("person1: " + p1);
        System.out.println("person2: " + p2);
        System.out.println("age: " + age + "\tname: " + name + "\n");
    }
}
```

Figure 3: ChangingPeople.java

```java
package lab_4;

import java.util.Date;

/**
 * An example generic class that compares three Comparable types and returns the
 * largest.
 *
 * @author Brandon Doyle
 * @param <T> A type that implements the ‘Comparable‘ interface. This generics
 *            statement basically means that <T is comparable with other instances
 *            of T>.
 */
public class Compare3<T extends Comparable<T>>
{
    /**
     * Find the largest of three arguments. This method is _not_ static because that
     * would not allow us to use non-static generic types.
     *
     * @param one   A Comparable object.
     * @param two   Another Comparable object.
     * @param three Yet another Comparable object.
     * @return The largest argument.
     */
    public T largest(T one, T two, T three)
    {
        T tmp = (one.compareTo(two) < 0) ? two : one;
        T max = (tmp.compareTo(three) < 0) ? three : tmp;

        return max;
    }

    /**
     * A static method that determines the largest of three Integers.
     *
     * @param x An integer.
     * @param y Another integer.
     * @param z Yet another integer.
     * @return The largest integer.
     */
    public static Integer largest(Integer x, Integer y, Integer z)
    {
        Integer tmp = (x.compareTo(y) < 0) ? y : x;
        Integer max = (tmp.compareTo(z) < 0) ? z : tmp;

        return max;
    }
}
```

Figure 4: Compare3.java

```java
package lab_4;

import java.util.Date;

public class Comparisons
{
    public static void main(String[] args) throws InterruptedException
    {
        var date1 = new Date();
        Thread.sleep(1000);
        var date2 = new Date();
        Thread.sleep(1024);
        var date3 = new Date();
        System.out.println(date3);
        var anotherInst = new Compare3<Date>();

        // The following should always print the same as ‘date3‘.
        System.out.println(anotherInst.largest(date1, date2, date3));

        // Now try our static method
        System.out.printf("Largest: %d\n", Compare3.largest(8, 12, 5));
    }
}
```

Figure 5: Comparisons.java