**Pre-Exam 1 Exercises**

This exercise is for your own preparation for Exam 1. It will not count towards your grade and is not for submission. An answer sheet will be provided to the exercise during the Exam 1 review two days before the exam day.

**Classes in Java**

1. (a) Complete each of the methods of the FeetAndInches class below,

public class FeetAndInches {

private int feet;

private int inches; // 0 <= inches < 12

// Alternate Constructor

// -- Sets feet and inches to values passed

public FeetAndInches(int feet, int inches) {

this.feet = feet;

this.inches = inches;

}

// Alternate Constructor

// -- Sets feet and inches from provided total inches

public FeetAndInches(int total\_inches) {

feet = total\_inches / 12;

inches = total\_inches % 12;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Copy Constructor

// -- Sets to value in object passed

public FeetAndInches(FeetAndInches other) {

//this.FeetAndInches = other;

feet = other.getFeet();

inches = other.getInches();

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Getters and Setters

public int getFeet() {

return feet;

}

public int getInches() {

return inches;

}

public void setFeet(int feet) {

this.feet = feet;

}

public void setInches(int inches) {

this.inches = inches;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// toString Method

// -- returns string of the form "10 ft. 8 in."

public String toString() {

//System.out.println(getFeet() + “ft. ” + getInches() + “in.” );

return feet + “ ft. “ + inches + “ins. ”;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// equals Method

public boolean equals(FeetAndInches other) {

//if ( this.feet == other.feet && this.inches == other.inches){

//return true;

//} else {

//return false;

//}

return feet == other.getFeet() && inches == other.getInches();

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// convert Method

// -- returns feet and inches converted to all inches

public int convertToInches() {

//return (this.getFeet() \* 12 + this.getInches());

return feet \* 12 + inches;

}

// add Method

// -- returns result such that 0 <= inches < 12

public FeetAndInches add(FeetAndInches other) {

//return this.convertToInches + other.convertToInches;

Int total\_feet = feet + other.getFeet();

Int total\_inches = inches + other.getInches();

If(total\_inches > 12) {

Total\_feet = total\_feet + 1;

Total\_inches = total\_inches – 12;

}

Return new FeetAndInches(total\_feet, total\_inches);

}

}

(b) For the FeetAndInches class above,

* Give a statement that creates a FeetAndInches object named meas1 with the  
  value 10 feet and 4 inches.

FeetAndInches meas1 = new FeetAndInches(10, 4);

* Give a statement that creates a copy of object meas1, named meas2.

FeetAndInches meas2 = new FeetAndInches(meas1);

* Give a statement that displays the value of FeetAndInches object meas1,

“The value of meas1 = . . .”

by use of the toString method, WITHOUT explicitly calling the toString method.

System.out.println(“The value of meas1 = “ + meas1 );

**Composition in Java**

2. Complete each of the methods of the Person class below as specified.

public class Person {

private String name;

private FeetAndInches height;

private int weight;

// Alternate Constructor

// -- Sets name, height and weight of person

// -- Height passed as total inches (e.g., 68)

public Person(String name, int height\_in\_inches, int weight) {

this.name = name;

this.weight = weight;

height = new FeetAndInches(height\_in\_inches);

}

// Getter Methods

public String getName() {

return name;

}

// -- Returns height as string of the form "5 ft. 8 in."

public String getHeight() {

return height.toString();

}

// -- Returns weight as string of the form "120 lbs."

public String getWeight() {

return (weight + “ lbs.”);

}

// -- Returns string of the form “John Smith, 5 ft. 8 in., 140 lbs.”

// -- MUST MAKE USE of toString of the FeetAndInches class

public String toString() {

return (name + “, ” + height + “, “ + getWeight());

}

}

}

**Subclasses in Java**

3. Assume that the following Cake class is implemented. Create a subclass named BirthdayCake that  
 in addition to the cake and icing type, also has a number of candles. Include in the BirthdayCake  
 class,

instance variables – as needed

a constructor – that is passed a cake type, icing type, and number of candles

getColor – that returns the number of candles on the cake

toString – that returns a string of the form "white cake, chocolate icing, 10 candles”

**NOTE**: The toString method of the BirthdayCake class must be implemented to  
 make use of the toString method of the Cake class.

public class Cake {

private String cake\_type; // white or chocolate

private String icing\_type; // butter\_cream or chocolate

// Alternate Constructor (sets cake\_type and icing\_type to values passed)

public XYCoord(String cake\_type, String icing\_type){ } // assume implemented

// Getters

public String getCakeType() { } // assume implemented

public int getIcingType { } // assume implemented

// toString Method (returns string of the form “white cake, chocolate icing”

public String toString() { } // assume implemented

}

public class BirthdayCake extends Cake{

private int candles;

public BirthdayCake(String cake\_type, String icing\_type, int candles){

super(cake\_type, icing\_type);

this.candles = candles;

}

public int getCandles {

return candles;

}

public String toString {

//return (Cake.toString() + getCandles() + “candles”);

Return super.toString() + “, “ + candles + “ candles”;

}

}

**Polymorphism in Java**

4. Suppose there is an abstract Line class as given below. Suppose there are also three subclasses  
 (subtypes) of the Line class – SolidLine, DashedLine and DottedLine. Complete the code below for the  
 underlined sections only that demonstrates the use of polymorphism for these types. (10 pts.)

public abstract class Line {

private XYCoord coord1;

private XYCoord coord2;

// Alternate Constructor

public Line(XYCoord coord1, XYCoord coord2) { } // assume implemented

// Draw Method

public abstract void draw();

}

// Main (demonstrating polymorphism)

Scanner input = new Scanner(System.in);

// variable declaration

int selection;

Line line;

// prompt user for line type

System.out.println(“Enter 1 – solid line, 2 – dashed line, 3 – dotted line: “);

selection = input.nextInt();

switch(selection) {

case 1: line = new soLine(); break;

case 2: line = new dashedLine(); break;

case 3: line = new dottedLine(); break;

case 4: line = new solidLine(); break;

}

// draws the selected line type

line.draw();