Computer Science 2
Fort Atkinson High School
Course Outline

"Introduction to Computer Programming in Java"

Unit 1: Introduction to Java

Unit 2: Using Objects

Unit 3: Implementing Classes

Unit 4: Fundamental Data Types

Unit 5: Decisions

Unit 6: Iteration

Unit 7: Arrays and Array Lists

Unit 8: Designing Classes

Unit 9: Sorting and Searching

Unit 1: *Introduction to Java*

Purpose:
The purpose of this unit is to familiarize the student with the Java language and the IDE used in the course. We will briefly explore the history of computer programming languages and create a very simple Java program. We will also take a look at common syntax errors.

Vocabulary

• CPU
• JVM
• High-level language
• Machine code
• Compiler
• Console Screen
• Library
• Case sensitive
• Comments
• Classes
• Methods
• String
• Syntax error
• Logic error
• Editor
• Source Code

Problem Set #1

- **Explain the difference between using a computer program and programming a computer.**
- **What is an Integrated Development Environment?**
- **What is the Java virtual machine?**
- **What is a compiler?**
- **Explain the difference between a syntax error and a logic error.**

Applications

App 1.1: NamePrinter

Write a application that displays your name inside a box on the console screen, like this:

```
+---------+
|  Dr. J  |
+---------+
```

App 1.2: FacePrinter

Write an application that prints a face, using text characters, like this:

```
  \\\\///
  | o o |
 ( |  .  | )
  | <> |
  --------
```

**Use comments to identify the code that creates each part of the face.

App 1.3: DialogViewer

Type in, compile, and run the following program:

```
import javax.swing.JOptionPane;

public class DialogViewer
{
        public static void main(String[] args)
        {
            JOptionPane.showMessageDialog(null, "Hello, World!");
            System.exit(0);
        }
}
```

Then ***modify*** the program to show the message "Hello, *your name*!"

Problem Set #1

Name _____ Hr ____

Introduction to Java

1. Explain the difference between using a computer program and programming a computer.

2. What is an Integrated Development Environment?

3. What is the Java virtual machine?

4. What is a compiler?

5. Explain the difference between a syntax error and a logic error.

Unit 2: *Using Objects*

Purpose:

The purpose of this unit is to learn how the computer uses variables, classes and methods to complete tasks. We will learn how to use objects that have been made by other programmers. We will learn how to test simple programs and learn the java vocabulary. In the graphics track, we will learn how to write programs that display simple shapes.

Vocabulary

• Identifiers
• String
• Variables
• Primitive Type
• int
• double
• boolean
• Classes
• Objects
• Methods
• Constructors
• Parameters
• Accessor methods
• Mutator methods
• Return values
• Void
• API documentation
• import
• Objects references
• Frames
• Components

Problem Set #2

- Explain the difference between an object and a class.
- Explain the difference between an object and an object reference.
- Explain the difference between the = symbol in Java and in mathematics.
- Define a variable to hold your first name and assign it your first name. Use the standard Java style convention.
- The Random class is defined in the java.util package. What do you need to do in order to use that class in your program?

Applications

App 2.1: AreaRectangle

Write an application that constructs a Rectangle object from the `Rectangle` class and then computes and prints its area. Use the import statement:

```
import java.awt.Rectangle;
```

Assign the rectangle a width and length. Use the getWidth and getHeight methods from the `Rectangle` class to access the width and length. Declare a variable for the Area and compute its value. Print the width, length and area. Also, in order to test the program, print the expected answer.

Major Hint:

```
import java.awt.Rectangle;

public class AreaRectangle
{
  public static void main(String[] args)
  {
    Rectangle myRectangle = new Rectangle(10,20);

    double width = myRectangle.getWidth();

    ……

    double area = width * length;
    System.out.println(width);

    ……..

  }
}
```

App 2.2: PerimeterRectangle

Write an application that constructs a Rectangle object from the `Rectangle` class and then computes and prints its perimeter.
Use the import statement:

import java.awt.Rectangle;.

Assign the rectangle a width and a height. Use the getWidth() and getHeight() methods from the `Rectangle` class to access the width and height. Create a variable for the perimeter and compute the perimeter using the width and length. Print the width, length, and perimeter. Also, in order to test the program, print the expected answer.

App 2.3: DieSimulator

Write an application that uses the Random class to simulate the cast of a die, print a random number between 1 and 6 every time that the program is run.

Note: The `Random` class (java.util.Random) implements a *random number generator*, which produces sequences of numbers that appear to be random. To generate random integers, you construct an object of the `Random` class, and then apply the nextInt method.
For example, the call: int myRandomNumber = generator.nextInt(5);

assigns a random number between 0 and 4 to the integer variable myRandomNumber.

Hint:

```
//  importing the Random Class
    import java.util.Random;
                    .
                    .
                    .
   //  This statement instantiates a new object from the Random class called "generator"
   Random generator = new Random();

   //  This statement creates a variable called "myRandomNumber" and assigns it a
   // randomly generated number by using the Random method called "nextInt()"
    int myRandomNumber = generator.nextInt(5);

    System.out.println(myRandomNumber)
```

App 2.4: LotteryPrinter

Write an application that picks a combination of numbers that the user should play in a lottery. In this lottery, players can choose 6 numbers (possibly repeated) between 1 and 49. (In a real lottery, repetitions aren't allowed, but we haven't yet discussed the programming constructs that would be required to deal with that problem.)
Your program should generate the 6 numbers and then print out a sentence such as "Play this combination – it'll make you rich!!", followed by a lottery combination.

App 2.5: ReplaceTester

Write an application that encodes a string by replacing all letters "i" with "!" and all letters "s" with "$". Use the replace method. Demonstrate that you can correctly encode the string "Mississippi". Print both the actual and expected result. **Hint**: This is a very short program.

App 2.6: HollePrinter:

Write an application that switches the letters "e" and "o" in a string use the replace method repeatedly. Demonstrate that the string "Hello, World!" turns into "Holle, Werld!" **Hint**: This program is a little trickier than it first appears.

App 2.7: AreaDialogBox

Type in the following program. *Then* modify it so that it outputs the perimeter.

```
import javax.swing.JOptionPane;

public class AreaProgram
{
  public static void main(String[] args)
  {
    String input = JOptionPane.showInputDialog("Enter length of rectangle ");
    double length = Double.parseDouble(input);
    input = JOptionPane.showInputDialog("Enter width of rectangle ");
    double width = Double.parseDouble(input);
    double area = length * width;
    JOptionPane.showMessageDialog(null, "Length = " + length + "\n" + "Width = " +
        width + "\n" + "Area = " + area);
    System.exit(0);
  }
}
```

Bonus Graphics App 1: The Alien Face

Type in the following two classes. After compiling each, run the FaceViewer. After it runs, modify the face, text, colors and frame size.

Class #1

```
import javax.swing.JFrame;

public class FaceViewer
{
  //  This is the driver program for the RectangleComponent Class
  public static void main(String[] args)
  {
    JFrame frame = new JFrame();
    frame.setSize(300, 400);
    frame.setTitle("An Alien Face");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    FaceComponent component = new FaceComponent();
    frame.add(component);

    frame.setVisible(true);

  }
}
```

Class #2

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Line2D;
import javax.swing.JPanel;
import javax.swing.JComponent;

/**
 * This component draws an alien face
 */

public class FaceComponent extends JComponent
{
  public void paintComponent(Graphics g)
  {
    //Recover Graphics2D
    Graphics2D g2 = (Graphics2D) g;

    //  Draw the head
    Ellipse2D.Double head = new Ellipse2D.Double(5, 10, 100, 150);
    g2.draw(head);

    //  Draw the eyes
    Line2D.Double eye1 = new Line2D.Double(25, 70, 45, 90);
    g2.draw(eye1);

    //  Draw the eyes
    Line2D.Double eye2 = new Line2D.Double(85, 70, 65, 90);
    g2.draw(eye2);

    //  Draw the mouth
    Rectangle mouth = new Rectangle(30, 130, 50, 5);
    g2.setColor(Color.RED);
    g2.fill(mouth);

    //  Draw the greeting
    g2.setColor(Color.BLUE);;
    g2.drawString("Hello, World!", 20, 190);
  }  //  end paintComponent
}  //  end FaceComponent
```

Problem Set #2

Name _____ Hr ____

Using Objects

1. Explain the difference between an object and a class.

2. Explain the difference between an object and an object reference.

3. Explain the difference between the = symbol in Java and in mathematics.

4. Define a variable to hold your first name and assign it the value of your first name. Use normal Java style conventions.

5. The Random class is defined in the java.util package. What do you need to do in order to use that class in your program?

Unit 3: *Implementing Classes*

Purpose:

The purpose of this unit is to become familiar with the process of implementing classes. We will learn how to implement simple methods, understand the purpose and use of constructors, and understand how to access instance fields and local variables. We will also learn to appreciate the importance of documentation comments. In the graphics track, we will learn how to implement classes for drawing graphical shapes.

Vocabulary

• Class methods
• Method definition
• Access specifier
• Return type
• Parameters
• Constructors
• Documentation comments
• javadoc
• Objects
• Instance fields
• Encapsulation
• "Black Box"
• Tester Class
• Implicit Parameter
• This

Problem Set #3

- **Explain the difference between a local variable and a parameter.**
- **Explain the difference between an instance field and a local variable.**
- **An example of a "black box" is the System.out, which is used to cause output to appear on the screen. What does that mean?**
- **Explain what a tester class is.**
- **What does it mean to have a void Return type?**

Applications

App 3.1: BankAccountTester

Write a class whose main method constructs a bank account, deposits $1,000, withdraws $500, withdraws another $400, and then prints the remaining balance. Also print the expected result.

Note: The *skeleton* of both the BankAccount class and the BankAccountTester class are provided for you.

Modify this Tester class for App 3.1

```
/**
   Tests the bank account class.
*/
public class BankAccountTester
{
  public static void main(String[] args)
  {
      BankAccount drjsAccount = new BankAccount();

      drjsAccount.deposit(1000);
      drjsAccount.withdraw(. . .);
      drjsAccount.withdraw(. . .);

      System.out.println(drsAccount.getBalance());
      System.out.println("Expected: . . .");
  }
}
```

BankAccount.java source code

```
/**
   A bank account has a balance that can be changed by
   deposits and withdrawals.
*/
public class BankAccount
{
   /**
      This Constructor creates a bank account with a zero balance.
   */
   public BankAccount()
   {
      balance = 0;
   }

   /**
      This Constructor creates a bank account with a given balance.
      @param initialBalance the initial balance
   */
   public BankAccount(double initialBalance)
   {
      balance = initialBalance;
   }

   /**
      This method Deposits money into the bank account.
      @param amount the amount to deposit
   */
   public void deposit(double amount)
   {
      double newBalance = balance + amount;
      balance = newBalance;
   }

   /**
      This method Withdraws money from the bank account.
      @param amount the amount to withdraw
   */
   public void withdraw(double amount)
   {
      double newBalance = balance - amount;
      balance = newBalance;
   }

   /**
      This method Gets the current balance of the bank account.
      @return the current balance
   */
   public double getBalance()
   {
      return balance;
   }

   private double balance;  //This is the current balance
}
```

App 3.2: Adding a method

Add the method: public void addInterest(double rate) to the BankAccount class that adds interest to the account balance using the given rate. The result of adding interest to the account is that the account has a new balance made from the old balance plus the interest.

For example, after the statements below the balance in momsSavings is $1,100.

```
BankAccount momsSavings = new BankAccount(1000);
momsSavings.addInterest(10);  //  10% interest
```

(Note: interest = principal x rate x time, or in this case, Interest = 1000 x .10 x 1). Also, modify the BankAccountTester class to print the actual and expected balance.

Modify this Tester class for App 3.2

```
/**
   Tests the bank account class.
*/
public class BankAccountTester
{
  public static void main(String[] args)
  {
      BankAccount drjsAccount = new BankAccount();

      drjsAccount.deposit(1000);
      drjsAccount.withdraw(. . .);
      drjsAccount.withdraw(. . .);

      drjsAccount.addInterest(10);

      System.out.println(. . .);
      System.out.println("Expected: . . .");
  }
}
```

Add this method to the bankAccount class and modify it for App 3.2

```
   /**
      . . .
   */
   public void addInterest(double rate)
   {
      . . .
   }
```

App 3.3: Adding a Constructor

Write a new Constructor for the BankAcount class, and add the instance field *interest*.

- Create a constructor that sets both the initial balance and the interest rate. This means that this constructor will have 2 parameters.
- You will need to create an instance field called: interestRate
- You will need to assign the parameter values to the instance fields (balance and interestRate)
- Remember that your parameter names and instance field names should be similar but NOT the same name.
- Modify your BankAccountTester class so that it constructs a bank account with an initial balance of $2,000 and an interest rate of 9%. Then add $500 to the balance, withdraw $100, add interest, then get the current balance. Print the current balance and print the expected value.

Major Hint: Add this constructor to the BankAccount class for App 3.3

```
//  This is the main constructor
public BankAccount()
 {
   balance = 0;
   interestRate = 0;
 }

 /**
   This Constructor creates a bank account with a given balance.
   @param initialBalance is the initial balance
   @param is the interest rate in percent
 */
 public BankAccount(double initialBalance, double rate)
 {
   balance = initialBalance;
   interestRate = rate;
 }

   …..

   private double interestRate;

   …..
```

App 3.4: BasketballPlayer

Create a class called **BasketballPlayer**. Every player has a name and total points. Supply appropriate constructor and methods

- getName()
- addPoints(int points)
- getTotalPoints()

You should have the following instance fields:

- playerName
- totalPoints

Supply a BasketballPlayerTester class that tests all methods.

Modify the following class as your tester class:

```
/**
   This program tests the BasketballPlayer class.
*/
public class BasketballPlayerTester
{
   public static void main(String[] args)
   {
      BasketballPlayer myPlayer = new BasketballPlayer("Tommy");

      // Add points for one game
      myPlayer.addPoints(12);

      // Add points for another game
      // Add points for another game
            .
            .
            .
      // Print actual and expected name, total points,
      // and average points per game
      System.out.println(myPlayer.getName());
            .
            .
            .

  }
}
```

Modify this class for App 3.4

```
/**
   A Basketball player who is scoring points.
*/
public class BasketballPlayer
{
   /**
      This Constructor creates a basketball player with a given name.
      @param n the name
   */
   public BasketballPlayer(String n)
   {
      . . .
   }

   /**
      Gets the name of this basketball player.
      @return the name
   */
   public String getName()
   {
      . . .
   }

   /**
      Adds the points for one game to the total.
      @param points, the points to add
   */
   public void addPoints(int points)
   {
      . . .
   }

   /**
      Gets the total of all points scored by the player
      @return the total points
   */
   public double getTotalPoints()
   {
      . . .
   }

   . . .
}
```

App 3.5: Mofifying the BasketballPlayer Class

Modify the class called **BasketballPlayer**. Every player has a name and total points. Now, add the code necessary to get the average points per game.

New Methods:

- getAveragePoints()

To compute the average points, you will need to keep track of the *number of games* that the basketball player played. To do this, create an instance field called *gamesPlayed* that is incremented every time the addPoints() method is called.

New Instance Fields:

- gamesPlayed

Modify the BasketballPlayerTester class that tests all methods and prints the expected result.

Hint: Modify this method to compute and get the average points per game for any object from the BasketballPlayer class.

```
/**
    Gets the average points per game.
    @return the average points per game
*/
public double getAveragePoints()
{
   . . .
}
```

App 3.6: VotingMachine

Implement a VotingMachine class that can be used for a simple election. Have methods to clear the machine state, to vote for a Democrat, to vote for a Republican, and to get the tallies for both parties.

Use the following class as your main class:

```
/**
   This program simulates an election.
*/
public class VotingSimulation
{
   public static void main(String[] args)
   {
      VotingMachine vm = new VotingMachine();
      vm.clear();

      vm.voteForDemocrat();
      vm.voteForRepublican();
      vm.voteForDemocrat();
      vm.voteForRepublican();
      vm.voteForRepublican();

      System.out.print("Democrats: ");
      System.out.println(vm.getDemocratVotes());
      System.out.print("Republicans: ");
      System.out.println(vm.getRepublicanVotes());
   }
}
```

Bonus Graphics App 2: Olympic Rings

Write a program that displays the Olympic rings. Color the rings in the Olympic colors. Provide a class OlympicRingViewer and a class OlympicRingComponent.

Use the following class as your main class:

```
import javax.swing.JFrame;

public class OlympicRingViewer
{
   public static void main(String[] args)
   {
      JFrame frame = new JFrame();

      final int FRAME_WIDTH = 300;
      final int FRAME_HEIGHT = 230;

      frame.setSize(300, 230);
      frame.setTitle("OlympicRingViewer");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      OlympicRingComponent component = new OlympicRingComponent();
      frame.add(component);

      frame.setVisible(true);
   }
}
```

Complete the following classes in your solution:

```
import javax.swing.JComponent;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Color;

/**
   Draws the olympic rings.
 */
public class OlympicRingComponent extends JComponent
{
   public void paintComponent(Graphics g)
   {
      Graphics2D g2 = (Graphics2D) g;

      // construct and draw five Ring objects
   }
}
```

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.Ellipse2D;

/**
   A class that draw the Olympic rings.
*/
public class Ring
{
   /**
      Constructs a circle that represents the Olympic rings.
      @param anX the x coordinate
      @param aY the y coordinate
      @param aRadius the radius of the circle
      @param aColor the color of the ring
   */
   public Ring(double anX, double aY, double aRadius, Color aColor)
   {
      . . .
   }
   /**
      Draws the ring.
      @param g2 the graphic context
   */
   public void draw(Graphics2D g2)
   {
      . . .
   }
   . . .
}
```

Problem Set #3

Name _____ Hr ____

Implementing Classes

1. Explain the difference between a local variable and a parameter.

2. Explain the difference between an instance field and a local variable.

3. An example of a “black box” is the System.out, which is used to cause output to appear on the screen. What does that mean?

4. Explain what a tester class is.

5. What does it mean to have a void Return type?

Unit 4: *Fundamental Data Types*

Purpose:

The purpose of this unit is to learn how to manipulate numbers and character strings. We will learn how to read program input and produce formatted output. We will learn about the numeric and string types such as integer, floating-point and String. We will also learn the role of constants.

Vocabulary

• Primitive types
• Integer
• Floating-point
• Overflow error
• Rounding errors
• Casting
• Integer division
• Final variables (Constants)
• Magic numbers
• The ”=” operator
• ++ and - -
• % operator
• The Math class
• Static methods
• Concatenation
• The “+” operator
• Parsing
• Substring
• Scanner class
• Escape sequences
• Formatting numbers
• Dialog Boxes for input and output

Problem Set #4

- Java has eight primitive types, including four integer types and two floating-point types. What does it mean to be a primitive type?
- Explain how the % operator works.
- If an input string contains the digits of a number, how do you go about obtaining the number value?
- What does it mean to have an overflow error?
- What does "integer division" mean? How can you override integer division?

Applications

App 4.1: CashRegister

Enhance the CashRegister class by adding separate methods

- enterDollars
- enterQuarters
- enterDimes
- enterNickels
- enterPennies

For *this* CashRegister, we will assume that user will never enter a less payment than the purchase price.

Modify this tester class:

```
public class CashRegisterTester
{
   public static void main (String[] args)
   {
      CashRegister myRegister = new CashRegister();
      myRegister.recordPurchase(20.37);
      myRegister.enterDollars(20);
      myRegister.enterQuarters(2);
        .
        .
      System.out.println("Change: " + myRegister.giveChange());
      System.out.println("Expected: 0.13");
   }
}
```

Modify the following class in your solution:

```
/**
   A cash register totals up sales and computes change due.
*/
public class CashRegister
{
   /**
      Constructs a cash register with no money in it.
   */
   public CashRegister()
   {
      purchase = 0;
      payment = 0;
   }
/**
   Records the sale of an item.
   @param amount the price of the item
*/
public void recordPurchase(double amount)
{
   double newTotal = purchase + amount;
   purchase = newTotal;
}

/**
   Computes the change due and resets the machine for the next customer.
   @return the change due to the customer
*/
public double giveChange()
{
   double change = payment - purchase;
   purchase = 0;
   payment = 0;
   return change;
}

public void enterDollars(int dollars)
{
   . . .
}

public void enterQuarters(int quarters)
{
   . . .
}

public void enterDimes(int dimes)
{
   . . .
}

public void enterNickels(int nickels)
{
   . . .
}

public void enterPennies(int pennies)
{
   . . .
}

private double purchase;
private double payment;

public static final double QUARTER_VALUE = 0.25;
public static final double DIME_VALUE = 0.1;
public static final double NICKEL_VALUE = 0.05;
public static final double PENNY_VALUE = 0.01;
}
```

App 4.2: A Counting Cash Register

Enhance the CashRegister class so that it keeps track of the total number of items in a sale. Count all recorded purchases and supply a method that returns the number of items of the current purchase. That is, every time the recordPurchase

method is called, increment the itemCount. Remember to reset the count at the end of the purchase.

New Method:

- int getItemCount()

New Instance Field:

- itemCount

Modify the CashRegisterTester tester class.

```
public int getItemCount()
{
   . . .
}
```

App 4.3: A Calculator

Write a program that prompts the user for two numbers, then prints

- The sum
- The difference
- The product
- The average
- The distance (The absolute value of the difference)
- The maximum (The larger of the two)
- The minimum (The smaller of the two)

Methods

- One for each of the items listed above

Instance Fields:

- firstNumber
- secondNumber

Then implement a class PairTester that constructs a Pair object, invokes its methods, and prints the results for all items listed above.

```
public class Pair
{
   /**
      This Constructs creates a pair of numbers.
      @param aFirst the first value of the pair
      @param aSecond the second value of the pair
   */
   public Pair(double aFirst, double aSecond)
         {
              . . .
         }

   /**
      This method computes the sum of the values of this pair.
      @return the sum of the first and second values
   */
   public double getSum()
       {
         double sum = firstNumber + secondNumber;
         return sum;
       }

            . . .
}
```

App 4.4: SodaCan

Implement a class SodaCan whose constructor receives the height and diameter of the soda can. Supply methods getVolume and getSurfaceArea. Supply a SodaCanTester class that tests your class.

App 4.5: QuadraticEquation

Implement a class QuadraticEquation whose constructor receives the coefficients a, b, c of the quadratic equation $ax^2 + bx + c = 0$. At this time, do not calculate the discriminant to determine how many solutions there are. (We will be doing that later.) Supply methods getSolution1 and getSolution2 that return the solutions, using the quadratic formula.

Methods:

- double getSolution1
- double getSolution2

Instance Fields:

- a
- b
- c

Write a test class QuadraticEquationTester that constructs a QuadraticEquation object, and prints the two solutions.

You may test your program using the equations:

$x^2 – x – 6 = 0$, which has solutions: x = 3 and x = -2
$x^2 – 25 = 0$, which has one solution x = 5 and x = 5

```
Hint:  QuadraticEquation eq1 = new QuadraticEquation(1, -1, -6);
```

Problem Set #4

Name _____ Hr ____

Fundamental Data Types

1. Java has eight primitive types, including four integer types and two floating-point types. What does it mean to be a primitive type?

2. Explain how the % operator works.

3. If an input string contains the digits of a number, how do you go about obtaining the number value?

4. What does it mean to have an overflow error?

5. What does “integer division” mean? How can you override integer division?

Unit 5: *Decisions*

Purpose:
The purpose of this unit is to understand how to implement decision-making into the programming process. We will learn how to compare integers, floating-point numbers, strings, and objects in order to change the flow of the program. This important concept is what allows a program to change depending on its values.

Vocabulary

• Decisions
• Condition
• If
• If-else
• Body of a statement
• Branching
• Block statement
• Relational operators
• Lexicographic comparison
• Null reference
• Sequences of comparisons
• Switch statement
• Nested branches
• “Dangling” else
• Enumerated types
• Boolean expressions
• Boolean operators
• && (And)
• || (Or)
• ! (Not)
• Boolean variables
• Test Coverage

Problem Set #5

- Of the following pairs of strings, which comes first in lexicographic order?
    a) “Tom”, “Dick”
    b) “Tom”, “Tomato”
    c) “church”, “Churchill”
    d) “car manufacturer”, “carburetor”
    e) “Harry”, “hairy”
- Explain the difference between an if/else if/else statement and nested if statements. Give an example for each.
- Explain the following terms, and give an example for each construct:
    a) Expression
    b) Condition
    c) Statement
    d) Simple statement
    e) Compound statement
    f) Block
- Explain why it is more difficult to compare floating-point numbers than integers. Write Java code to test whether an integer n equals 10 and whether a floating-point number x equals 10.
- Explain the difference between the == operator and the equals method when comparing strings.

Applications

App 5.1: Enhanced BankAccount

Enhance the BankAccount class of Chapter 3 by
Rejecting negative amounts in the deposit and withdrawal methods.
Rejecting withdrawals that would result in a negative balance.

App 5.2: Paycheck

Write a program that reads in the hourly wage of an employee. Then ask how many hours the employee worked in the past week. Be sure to accept partial hours. Compute the gross pay. Any overtime work (over 40 hours per week) is paid at 1.5 times the regular wage. Solve this problem by implementing a class paycheck.

Use the following class as your main class:

```
import java.util.Scanner;
```

App 5.3: Improving the Quadratic Equation Class

Modify the QuadraticEquation Class so that it tests for real and non-real solutions.

- If the *discriminant* $b^2 - 4ac$ is positive, display a message that there are two real solutions.
- If the *discriminant* $b^2 - 4ac$ is equal to zero, display a message that there is only one real solution.
- If the *discriminant* $b^2 - 4ac$ is negative, display a message stating that there are no real solutions.

Methods:

- double getDiscriminant which calculates and returns the value of the discriminant. This method may be called by the other methods in the class
- int getNumberOfSolutions() that returns the number of solutions
- void displayHowManySolutions() that diplays a message describing how many solutions the equation has as described above
- double getSolution1() that returns the smaller of the two solutions if there are two solutions
- double getSolution2() that returns the larger of the two solutions if there are two solutions

Instance Fields:

- a
- b
- c
- discriminant

Note: If there is only one solution, print *only* that solution.

Modify the following class as your tester class:

```
/**
   This program tests the QuadraticEquation class.
*/
public class QuadraticEquationTester
{
   public static void main(String[] args)
   {
      QuadraticEquation eq1 = new QuadraticEquation(2, 2, -4);

      System.out.println(eq1.getNumberOfSolutions)

      if (eq1.getNumberOfSolutions == 2)
        {
          System.out.println(eq1.getSolution1());
          System.out.println(eq1.getSolution2());
        }
      else
        {
          System.out.println(eq1.getSolution1());
        }


      QuadraticEquation eq2 = new QuadraticEquation(2, 2, 4);
            .
            .
            .

   }
}
```

App 5.4: Deck of Cards

Write a program that takes user input describing a playing card in the following shorthand notation:

Notation	Meaning
A	Ace
2 ... 10	Card values
J	Jack
Q	Queen
K	King
D	Diamonds
H	Hearts
S	Spades
C	Clubs

Your program should print the full description of the card. For example,

Enter the card notation:
4S
Four of spades

Implement a class Card whose constructor takes the card notation string and whose getDescription method returns a description of the card. If the notation string is not in the correct format, the getDescription method should return the string "Unknown". Use the following class as your main class:

import java. util.Scanner;

```
/**
   This is a test for the Card class, which outputs the full
   description of a deck of cards.
*/
public class CardPrinter
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);

      System.out.println("Enter the card notation:");
      String input = in.nextLine();
      Card myCard = new Card(input);
      System.out.println(myCard.getDescription());
   }
}
```

Problem Set #5 *Decisions*

Name _____ Hr ____

1. Of the following pairs of strings, which comes first in lexicographic order?
   a) “Tom”, “Dick”
   b) “Tom”, “Tomato”
   c) “church”, “Churchill”
   d) “car manufacturer”, “carburetor”
   e) “Harry”, “hairy”
2. Explain the difference between an if/else if/else statement and nested if statements. Give an example for each.

3. Explain the following terms, and give an example for each construct:
   a) Expression
   b) Condition
   c) Statement
   d) Simple statement
   e) Compound statement
   f) Block

4. Explain why it is more difficult to compare floating-point numbers than integers. Write Java code to test whether an integer n equals 10 and whether a floating-point number x equals 10.

5. Explain the difference between the == operator and the equals method when comparing strings.

Unit 6: *Iteration*

Purpose:
The purpose of this unit is to understand how looping works. We will learn the *while* and *for* statements. We will also learn how *nest* loops, process input and avoid infinite loops.

Vocabulary

• Iteration
• Loop
• While
• Do
• For
• "Off by One"
• Symmetric loop
• Asymmetric loop
• Spaghetti code
• Counter
• Nested Loop
• Sentinel value
• Random Numbers
• Simulations
• Debugger

Problem Set #6
Explain the difference between the *while* loop and the *for* loop.
What is an *off-by-one* error? How does a programmer avoid being *off-by-one*?
What does it mean for loops to be *nested*?
What is a debugger program? Explain.
What is an infinite loop? Which loop structure, *while* or *for,* is more susceptible to an infinite loop? Explain.

Applications

App 6.1: CurrencyConverter

Write a program that asks the user to enter today's exchange rate between U.S. dollars and the euro. Then the program reads the U.S. dollar value and converts it to euro values. Continue this until the user enters 'Q'.

Notes:

- The main program can be in the CurrencyConverter class
- Look up the conversion on the internet

**BONUS:* Create this program as a GUI application

App 6.2: RandomDataAnalyzer

Write a program that generates 100 random numbers between 0 and 1000. The program should add up all of the numbers, print the total and print the average.

Problem Set #6
Iteration

Name _____ Hr ____

1. Explain the difference between the *while* loop and the *for* loop.

2. What is an *off-by-one* error? How does a programmer avoid being *off-by-one*?

3. What does it mean for loops to be *nested*?

4. What is a debugger program? Explain.

5. What is an infinite loop? Which loop structure, *while* or *for*, is susceptible to an infinite loop? Explain.

Unit 7: *Arrays and Array Lists*

Purpose:
The purpose of this unit is to familiarize ourselves with the use of arrays and array lists. These constructs are used to keep track of a list of data. We learn how to fill an array, add data to and subtract data from an array lists. We will also study common array algorithms.

Vocabulary

• Array
• ArrayList
• Reference
• Array elements
• Index
• Length of an array
• Bounds error
• Initialization
• Enhanced for loop
• Finding a value
• Finding max or min
• Two-Dimensional arrays
• Copying arrays
• Parallel arrays
• Arrays of Objects

Problem Set #7
What is an array? Explain.
What is the first index value of every array?
What is the notation to access the *third* element of an array? Create an example that demonstrates your explanation.
What method would you use to find out how many elements are in an array?
What is the difference between an array and an arraylist? Explain.

Applications

App 7.1: MyFriends

Write an application that allows the user to add the names of 5 friends to their *friendlist*. After the user enters the names of the 5 friends, the program prints the list in order, then in reverse order.

App 7.2: HaveMoreFriends

Write an application similar to the App 7.1, however, the user is allowed to enter up to 20 friends to their *friendlist*. The user must enter at least one friend. Then, the user is allowed to quit entering names at any time. After the user enters the names of their friends, the program prints the list in order, then in reverse order.

App 7.3: CD

Write a class called *CD* that will be used to allow the user to create a list of their favorite CD's. Each CD should include the Band Name and Title of the CD. Create a class called *CDTester* that allows the user to enter as many CD's as they want. When the user is done entering CD's, the program prints a list of their CD's in format listed below.

Note: Be sure to take the time to design this application before you begin typing!

Example Output:

1. AC/DC – Back in Black
2. Van Halen – 1984
3. Queen – Queen II
4. Rush – Moving Pictures
5. Journey – Evolution

Problem Set #7

Name _____ Hr ____

Arrays and Array Lists

1. What is an array? Explain.

2. What is the first index value of every array?

3. What is the notation to access the *third* element of an array? Create an example that demonstrates your explanation.

4. What method would you use to find out how many elements are in an array?

5. What is the difference between an array and an arraylist? Explain.

Unit 8: *Designing Classes*

Purpose:
The purpose of this unit is to learn how to create and choose appropriate classes to implement. We will carefully analyze the concepts of cohesion and coupling and learn how to minimize the use of side effects. We will take a closer look at the responsibilities of methods and their callers in terms of preconditions and postconditions. We will learn the difference between instance methods and static methods and introduce the concept of static fields while understanding the scope rules for local variables and instance fields.

Vocabulary

• Single Concept
• Cohesion
• Coupling
• Consistency
• Accessor method
• Mutators method
• Immutable class
• Side effects
• Precondition
• Postcondition
• Static method
• Static Field
• Scope
• Local variables
• Packages

Problem Set #8
"A class should represent a single concept." Explain what this means.
Explain the concept of cohesiveness.
What is a *side effect*? Explain.
Compare and contrast the idea of a *precondition* and a *postcondition*.
What is a package in Java?

Applications

App 8.1: Payroll Department

Implement a program that prints paychecks for a group of student workers for the School District of Fort Atkinson. Your program should prompt the user for the name, hourly wage, and hours worked for student. Deduct federal (8%), state (6%), and Social Security taxes(7.65%). The program should print all pertinent data in a table format.

Example Output:

Name: Tommy Johnson
Hours Worked: 10
Hourly Wage: 8.75
Gross Pay: 87.50
Fed Tax: 7.00
State Tax: 5.25
SS Tax: 6.69
Total Deductions: 18.94
Net Pay: 68.56

You ***must*** include:

- Accessor methods for each instance field
- Mutator methods for each instance field

*Bonus Options

- Make this program a GUI
- Make this program so that it allow the user to make a list of several employees.

Problem Set #8

Name _____ Hr ____

Designing Classes

1. “A class should represent a single concept.” Explain what this means.

2. Explain the concept of cohesiveness.

3. What is a *side effect*? Explain.

4. Compare and contrast the idea of a *precondition* and a *postcondition*.

5. What is a package in Java?

Unit 9: *Sorting and Searching*

Purpose:

One of the most common tasks in data processing is sorting. For example, a collection of athletes on a track team may be wanted to be sorted out in alphabetical order or sorted by their times. In this unit, we will study several sorting methods, learn how to implement them, and compare their performance.

Vocabulary

• Sorting algorithms
• Swapping
• Selection sort
• Insertion sort
• Merge sort
• Quicksort
• Searching
• Binary Search

Problem Set #9

What does it mean to *sort* a set of data?
Name four prominent sorting algorithms.
What does the term *swapping* refer to when sorting data?
What is the general algorithm behind the *Binary Search*?

Applications

App 9.1: Selection Sort Algorithm

Make a list of 5 integers. Use the *selection* sort algorithm to sort an array of integers in ascending order. Print the original list and sorted list.

App 9.2: Modified Selection Sort Algorithm

Modify App 9.1 so that the array is sorted first in ascending order, then in descending order. Print both lists.

App 9.3: Merge Sort Algorithm

Make a list of 5 integers. Use the *merge* sort algorithm to sort an array of integers in ascending order. Print the original list and sorted list.

App 9.4: Modified Merge Sort Algorithm

Modify App 9.3 so that the array is sorted first in ascending order, then in descending order. Print both lists.

Problem Set #9

Name _____ Hr ____

Sorting and Searching

1. What does it mean to *sort* a set of data?

2. Name four prominent sorting algorithms.

3. What does the term *swapping* refer to when sorting data?

4. What is the general algorithm behind the *Binary Search*?