Homework 3

(Due: April 18, 2023)

Question 1: 50 points

This question explores use of the **builder software design pattern** for the systematic assembly of objects having many attributes. We briefly discussed this software pattern in class along with the source code files in java-code-design-patterns/src/builder01/.

The proposed approach embeds a builder class inside the application class itself.

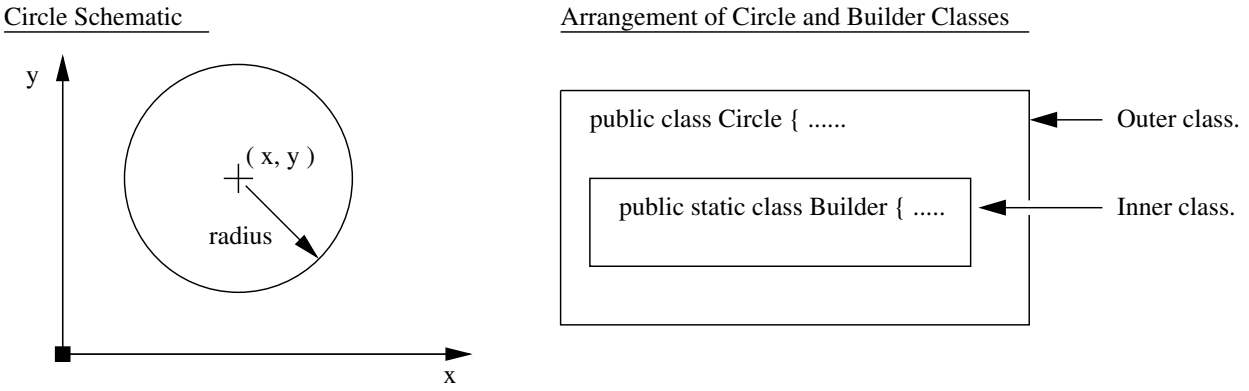

Figure 1: Schematic for circles having center (x,y) and radius r.

To see how the builder works on a simple example, the left-hand side of Figure 1 shows a circle defined by radius r and (x,y) coordinates at the center. The corresponding arrangement of outer (Circle) and inner (Builder) classes is shown on the right-hand side of Figure 1.

Notice that because the Builder class is defined as being static, there is no need to a create a Circle object before calling methods in the Builder. Access to the methods within the Builder is defined by our usual dot notation, i.e., Circle.Builder.(name of method)(args to method). Both of these concepts are employed in the test program.

File 1: Circle.java

source code

```
/*
```

```
 * ========================================================================
 * Circle.java: Circles have a name, a center (x,y), and a radius r.
 *
 * Additional parameters include distance units and a boolean for indicating
 * whether or not the circle should be filled.
 *
 * Written by: Mark Austin                                          May, 2019
 * ========================================================================
 */

package shapebuilder;

import java.util.*;
import java.lang.Math;

import unit.DistanceUnit;

public class Circle {
   private String         name;
   private double x, y, radius;
   private DistanceUnit unit;
   private boolean fill;

   // Circle builder class ...

   public static class Builder {

       // required parameter

       private String name;

       // optional parameters

       private double x, y, radius;
       private DistanceUnit unit = DistanceUnit.DEFAULT;
       private boolean fill = false;

       // Builder constructor method with required parameter ...

       public Builder( String name ) {
          this.name = name;
       }

       // Initialize optional parameters ..

       public Builder center ( double x, double y ) {
          this.x = x;
          this.y = y;
          return this;
       }

       public Builder radius ( double value ) {
          this.radius = value;
          return this;
       }
```

```
        public Builder units ( DistanceUnit value ) {
           this.unit = value;
           return this;
        }

        public Builder fill ( boolean fill ) {
           this.fill = fill;
           return this;
        }

        // Assemble the circle object ..

        public Circle build() {
           return new Circle( this );
        }
    }

    // Main constructor method ....

    public Circle( Builder builder ) {
       this.name   = builder.name;
       this.radius = builder.radius;
       this.x      = builder.x;
       this.y      = builder.y;
       this.unit   = builder.unit;
       this.fill   = builder.fill;
    }

    // Compute circle perimeter and area ...

    public double perimeter () {
       return 2.0*Math.PI*radius;
    }

    public double area () {
       return Math.PI*radius*radius;
    }

    // Create string representation of circle ...

    public String toString() {
       StringBuffer buf = new StringBuffer();

       String units = unit.toString();
       buf.append( String.format("Circle(%3s): distance units = %s, fill = %s\n", name, units, fill )
       buf.append( String.format("           : center (x,y) = (%.1f %s, %.1f %s)\n", x, units, y, unit
       buf.append( String.format("           : radius       = %.1f %s\n",         radius, units ));
       buf.append( String.format("           : perimeter    = %.1f %s\n",    perimeter(), units ));
       buf.append( String.format("           : area         = %.1f %s^2\n",       area(), units ));

       return buf.toString();
    }
}
```

File 2: DistanceUnit.java

source code

```
/*
 *  ============================================================================
 *  DistanceUnit.java: Define units for measuring distances, along with methods
 *  for units conversion.
 *
 *  Written by: Mark Austin                                                May, 2019
 *  ============================================================================
 */

package unit;

public enum DistanceUnit {
    INCH(0.0254, "in", "inches"),
    YARD(0.9144, "yd", "yards"),
    FEET(0.3048, "ft", "feet"),
    KILOMETERS(1000.0, "km", "kilometers"),
    MILLIMETERS(0.001, "mm", "millimeters"),
    CENTIMETERS(0.01, "cm", "centimeters"),
    MILES(1609.344, "miles"),
    METERS(1, "m", "meters");

    public static final DistanceUnit DEFAULT = METERS;
    private double meters;
    private final String[] names;

    DistanceUnit(double meters, String...names) {
        this.meters = meters;
        this.names  = names;
    }

    // Return string representation of distance unit ...

    public String toString() {
        return names[0];
    }
}
```

File 3: TestCircleBuilder.java

source code

```
/*
```

```
 *  ========================================================================
 *  TestCircleBuilder.java: Exercise Circle and Builder classes.
 *
 *  Written By: Mark Austin                                          May 2019
 *  ========================================================================
 */

package demo;

import shapebuilder.*;
import geometry.*;
import unit.*;

public class TestCircleBuilder {
   public static void main( String args[] ) {
   double dX, dY;

      System.out.printf("TestCircleBuilder program ...\n");
      System.out.printf("============================================================ \n");

      // Assemble test circle 1 ...

      DistanceUnit u01 = DistanceUnit.MILES;
      Circle circleA = new Circle.Builder("cA").units(u01).build();

      // Assemble test circle 2 ...

      DistanceUnit u02 = DistanceUnit.CENTIMETERS;
      Circle circleB = new Circle.Builder("cB").center(2.0,2.0).radius(5.0)
                                               .units(u02).fill(true).build();

      // Print details of circle assemblies ...

      System.out.println( circleA );
      System.out.println( circleB );

      System.out.printf("============================================================ \n");
      System.out.printf("Done! ... \n");
   }
}
```

The script of program input and output is as follows:

```
prompt >> ant geom03

     [java] TestCircleBuilder program ...
     [java] ============================================================
     [java] Circle( cA): distance units = miles, fill = false
     [java]            : center (x,y) = (0.0 miles, 0.0 miles)
     [java]            : radius       = 0.0 miles
     [java]            : perimeter    = 0.0 miles
```

```
     [java]            : area         = 0.0 miles^2
     [java]
     [java] Circle( cB): distance units = cm, fill = true
     [java]            : center (x,y) = (2.0 cm, 2.0 cm)
     [java]            : radius       = 5.0 cm
     [java]            : perimeter    = 31.4 cm
     [java]            : area         = 78.5 cm^2
     [java]
     [java] ============================================================
     [java] Done! ...

prompt >>
```

Please look at the source code carefully and answer the questions that follow:

[1a] (10 pts). Draw and label a diagram that shows the organizational arrangement of Java packages and user-defined source code files.

[1b] (10 pts). Draw and label a diagram that shows the relationship among the Circle, Builder, DistanceUnit and TestCircleBuilder classes.

[1c] (10 pts). Draw and label a diagram showing the layout of memory generated by the statements:

```
// Assemble test circle 1 ...

DistanceUnit u01 = DistanceUnit.MILES;
Circle circleA = new Circle.Builder("cA").units(u01).build();
```

[1d] (10 pts). Briefly explain the **sequence of events** that occurs when the statement

```
Circle circleB = new Circle.Builder("cB").center(2.0,2.0).radius(5.0)
                                          .units(u02).fill(true).build();
```

is executed.

[1e] (10 pts). The output for circle cB displays with distance units as cm, i.e.,

```
[java] Circle( cB): distance units = cm, fill = true
[java]            : center (x,y) = (2.0 cm, 2.0 cm)
[java]            : radius       = 5.0 cm
[java]            : perimeter    = 31.4 cm
[java]            : area         = 78.5 cm^2
```

Explain how the DistanceUnit class handles this detail (all reasonable answers will be accepted).

Question 2: 50 points

This question explores use of the **visitor software design pattern** to create textual descriptions of orders made at a ficticious online bookstore called Tiny-Amazon. The scenario for exercising the prototype software proceeds as follows: A young reader (i.e., Angela Austin) orders various quantities of children's books (i.e., The Cat in the Hat; Green Eggs and Ham; Hop on Pop) from Tiny-Amazon.

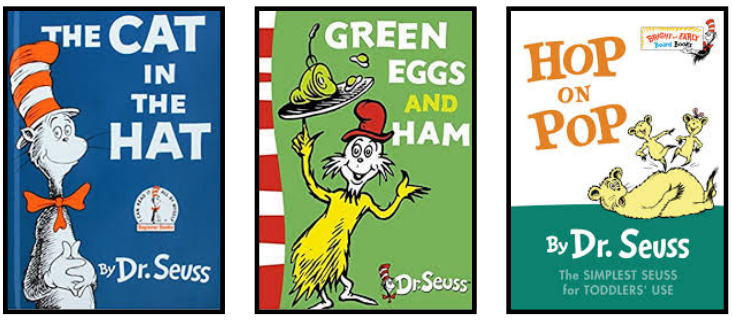

Figure 2: Books for children authored by Dr. Seuss.

Individual "order items" (i.e., book item times quantity) are added to a shopping cart. Printable descriptions of the order contain: (1) details on the customer name and ID number, (2) the cost for each item added to the shopping cart, and (3) the total cost for the complete order.

The source code for the software prototype is organized into eight files:

File: Visitor.java

source code

```
package amazon;

public interface Visitor {
   public void visit( ShoppingCartItem item );
}
```

File: Visitable.java

source code

```
package amazon;

public interface Visitable {
```

```
   public double       getPrice();
   public int       getQuantity();
   public String getDescription();
   public void accept( Visitor visitor );
}
```

File: Book.java

source code

```
/*
 * =====================================================================
 * Book.java: Simple model of a book ...
 * =====================================================================
 */

package amazon;

public class Book {
   private String   name;
   private String author;
   private double  price;

   // Constructor method ...

   public Book( String name, String author, double price ) {
      this.name   = name;
      this.author = author;
      this.price  = price;
   }

   // Retrieve data from book model ...

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getAuthor() {
      return author;
   }

   public void setAuthor(String author) {
      this.author = author;
   }

   public void setPrice( double price ) {
      this.price = price;
```

```
   }

   public double getPrice() {
      return price;
   }

   // Create string description of book ....

   public String toString() {
      String description = String.format(" Book: %s", getName() );
      return description;
   }
}
```

File: Customer.java

source code

```
/*
 *  ================================================================================
 *  Customer.java: Create a Customer object with an association to a shopping cart.
 *  ================================================================================
 */

package amazon;

public class Customer {
    private String customerName;  // Customer name ...
    private int      customerID;  // Customer id ...
    public ShoppingCart    cart;  // Customer's shopping cart

    // Constructors ....

    public Customer () {
       this.cart = null;
    }

    public Customer ( String customerName ) {
       this.customerName = customerName;
       this.cart = null;
    }

    // Attribute accessors

    public String getName() {
       return customerName;
    }

    public void setName(String customerName) {
       this.customerName = customerName;
    }
```

```
    public int getID() {
       return customerID;
    }

    public void setID( int customerID ) {
       this.customerID = customerID;
    }

    // Assign shopping cart to customer ...

    public ShoppingCart getShoppingCart() {
       return cart;
    }

    public void add ( ShoppingCart cart ) {
       if (cart != null)
           this.cart = cart;
    }
}
```

File: PrintOrderVisitor.java

source code

```
/*
 * ================================================================================
 * PrintOrderVisitor.java: Create printable version of items in shopping cart.
 * ================================================================================
 */

package amazon;

public class PrintOrderVisitor implements Visitor {
   StringBuffer order = new StringBuffer();
   private double totalPrice = 0.0;
   int itemNo = 1;

   // Visit items and compute price ...

   @Override
   public void visit( ShoppingCartItem item ) {
      String description = item.getDescription();
      int       quantity = item.getQuantity();
      double       price = item.getPrice();
      double    itemCost = quantity * price;

      totalPrice += quantity*price;

      order.append( String.format("Item %2d: %s ... \n", itemNo, description ) );
      order.append( String.format(" --   Quantity: %2d   \n", quantity ));
```

```
      order.append( String.format(" -- Unit price: $%5.2f \n", price ) );
      order.append( String.format(" --  Item cost: $%5.2f \n\n", itemCost ) );
      itemNo = itemNo + 1;
   }

   // Visit customer and add information ...

   public void appendCustomerInfo( Customer customer ) {
      order.append( String.format("================================== \n") );
      order.append( String.format("Customer Name: \"%s\" ... \n", customer.getName() ) );
      order.append( String.format("Customer   ID: \"%s\" ... \n", customer.getID() ) );
      order.append( String.format("================================== \n\n") );
   }

   // Append total cost to print order ....

   public void appendTotalCost() {
      order.append( String.format("============================== \n") );
      order.append( String.format("Total price = $%5.2f \n", totalPrice ) );
      order.append( String.format("============================== \n") );
   }

   // Retrieve the internal state ...

   public double getTotalPrice() {
      return totalPrice;
   }

   // Retrieve print order ...

   public String getPrintOrder() {
      return order.toString();
   }
}
```

File: ShoppingCartItem.java

source code

```
/*
 * ================================================================
 * ShoppingCartItem.java: Model an item in the shopping cart ...
 * ================================================================
 */

package amazon;

public class ShoppingCartItem implements Visitable {
   private Book    item;
   private int quantity;
```

```
   // Constructor method ...

   public ShoppingCartItem( Book item, int quantity ) {
      this.item     = item;
      this.quantity = quantity;
   }

   // Set/get methods ...

   public String getDescription() {
      return item.toString();
   }

   public double getPrice() {
      return item.getPrice();
   }

   public int getQuantity() {
      return quantity;
   }

   // Accept method for visitors ...

   @Override
   public void accept( Visitor visitor ) {
      visitor.visit( this );
   }
}
```

File: ShoppingCart.java

source code

```
/*
 *  ====================================================================
 *  ShoppingCart.java: Simple shopping cart model ...
 *  ====================================================================
 */

package amazon;

import java.util.ArrayList;
import java.util.List;

public class ShoppingCart {
   private ArrayList<Visitable> items;
   private Customer customer = null;

   // Constructor method ....
```

```
   public ShoppingCart() {
      items = new ArrayList<Visitable>();
   }

   // Assign customer to the shopping cart ...

   public void add( Customer customer ) {
      this.customer = customer;
   }

   // Add item to the shopping cart ...

   public void add( Visitable item ) {
      items.add ( item );
   }

   // Retrieve no items in shopping cart ...

   public int getNoItemsInCart() {
      return items.size();
   }

   // -----------------------------------------------------
   // Create printable version of items in shopping cart ...
   // -----------------------------------------------------

   public String getPrintOrder() {

      // Create print visitor ...

      PrintOrderVisitor visitor = new PrintOrderVisitor();

      // Add customer information to print visitor ...

      visitor.appendCustomerInfo( customer );

      // Iterate through items in shopping cart items ...

      for(Visitable item: items) {
          item.accept( visitor );
      }

      // Append total cost to the print order ...

      visitor.appendTotalCost();

      // Return string representation of shopping cart order ..

      String printOrder = visitor.getPrintOrder();
      return printOrder;
   }
}
```

File: TestAmazonShoppingCart.java

source code

```
/*
 *  ============================================================================
 *  TestAmazonShoppingCart.java: Simulation of a tiny-amazon shopping cart order.
 *
 *  Written by: Mark Austin                                                May 2018
 *  ============================================================================
 */

package demo;

import amazon.*;

public class TestAmazonShoppingCart {
    public static void main(String[] args) {

        // Part [a] -- Create small collection of children's books ...

        Book book01 = new Book( "The Cat in the Hat", "Dr Seuss", 10.0 );
        Book book02 = new Book( "Green Eggs and Ham", "Dr Seuss", 12.0 );
        Book book03 = new Book(         "Hop on Pop", "Dr Seuss", 12.0 );

        // Part [b] -- Create an empty shopping cart ....

        ShoppingCart cart01 = new ShoppingCart();

        // Part [c] -- Create shopping cart items ...

        ShoppingCartItem item01 = new ShoppingCartItem( book01, 1 );
        ShoppingCartItem item02 = new ShoppingCartItem( book02, 2 );
        ShoppingCartItem item03 = new ShoppingCartItem( book03, 1 );

        // Part [d] -- Add items to the shopping cart ...

        cart01.add( item01 );
        cart01.add( item02 );
        cart01.add( item03 );

        // Part [e] -- Customer (Angela Austin) logs into Amazon ...

        Customer angela01 = new Customer( "Angela Austin" );
        angela01.setID ( 12345 );

        // Part [f] -- Establish bi-directional association between the
        //          -- customer and the shopping cart ...

        angela01.add ( cart01 );
        cart01.add ( angela01 );

        // Part [g] -- Create printable version of shopping cart order ...
```

```
        System.out.printf("*** ================================================== \n" );
        System.out.printf("*** Printable Version of Tiny-Amazon Shopping Cart Order \n" );
        System.out.printf("*** ================================================== \n\n" );

        String printOrder = cart01.getPrintOrder();
        System.out.println( printOrder );

        System.out.printf("*** ================================================ \n" );
        System.out.printf("*** Finished !! \n" );
    }
}
```

The script of program input and output is as follows:

```
Script started on Mon May  7 15:58:36 2018
prompt >> ant run07
Buildfile: /Users/austin/ence688r.d/build.xml

compile:
run07:
  [java] *** ================================================
  [java] *** Printable Version of Tiny-Amazon Shopping Cart Order
  [java] *** ================================================
  [java]
  [java] ====================================
  [java] Customer Name: "Angela Austin" ...
  [java] Customer   ID: "12345" ...
  [java] ====================================
  [java]
  [java] Item  1:  Book: The Cat in the Hat ...
  [java]  --   Quantity:  1
  [java]  -- Unit price: $10.00
  [java]  --  Item cost: $10.00
  [java]
  [java] Item  2:  Book: Green Eggs and Ham ...
  [java]  --   Quantity:  2
  [java]  -- Unit price: $12.00
  [java]  --  Item cost: $24.00
  [java]
  [java] Item  3:  Book: Hop on Pop ...
  [java]  --   Quantity:  1
  [java]  -- Unit price: $12.00
  [java]  --  Item cost: $12.00
  [java]
  [java] ============================
  [java] Total price = $46.00
  [java] ============================
  [java]
  [java] *** ============================================
  [java] *** Finished !!
```

```
BUILD SUCCESSFUL
Total time: 1 second
prompt >> exit
Script done on Mon May  7 15:58:50 2018
```

[2a] (10 pts). Draw and label a diagram the shows the layout of memory generated by the four statements. Note, this is a subset of statements in parts [a] through [d] in the test program.

```
Book book01 = new Book( "The Cat in the Hat", "Dr Seuss", 10.0 );

ShoppingCart cart01 = new ShoppingCart();

ShoppingCartItem item01 = new ShoppingCartItem( book01, 1 );

cart01.add( item01 );
```

[2b] (10 pts). Draw and label a diagram for association relationship created by the statements in Parts [e] and [f] of the test program.

[2c]. (10 pts). Briefly explain how the `Visitor` and `Visitable` inferfaces are used in `ShoppingCartItem`.

[2d] (10 pts). Draw and label a diagram that shows how the interaction between the fragment of code:

source code

```
/*
 *  ====================================================================
 *  ShoppingCart.java: Simple shopping cart model ...
 *  ====================================================================
 */

public class ShoppingCart {
   private ArrayList<Visitable> items;

   ... lines of code removed ...

   public String getPrintOrder() {

      ... lines of code removed ...

      // Iterate through items in shopping cart items ...

      for(Visitable item: items) {
          item.accept( visitor );
      }

      ... lines of code removed ...
   }
}
```

in `ShoppingCart.java` and

```
public class ShoppingCartItem implements Visitable {
   private Book    item;
   private int quantity;

   ... code removed ...

   @Override
   public void accept( Visitor visitor ) {
      visitor.visit( this );
   }
}
```

works. All reasonable answers will be accepted.

Question 2d continued:

[2e]. (10 pts). The prototype software only works with books. Briefly indicate how you would extend the software infrastructure to work with books, video DVDs and music CDs. All reasonable answers will be accepted.