Bài 1: Cửa hàng café Starbuzz.

Cửa hàng có nhiều loại café, mỗi loại café có giá khác nhau. Ngoài ra có thể thêm các gia vị khác cho café. Giá của café được tính bằng giá của loại café cộng với giá của các loại gia vị thêm.

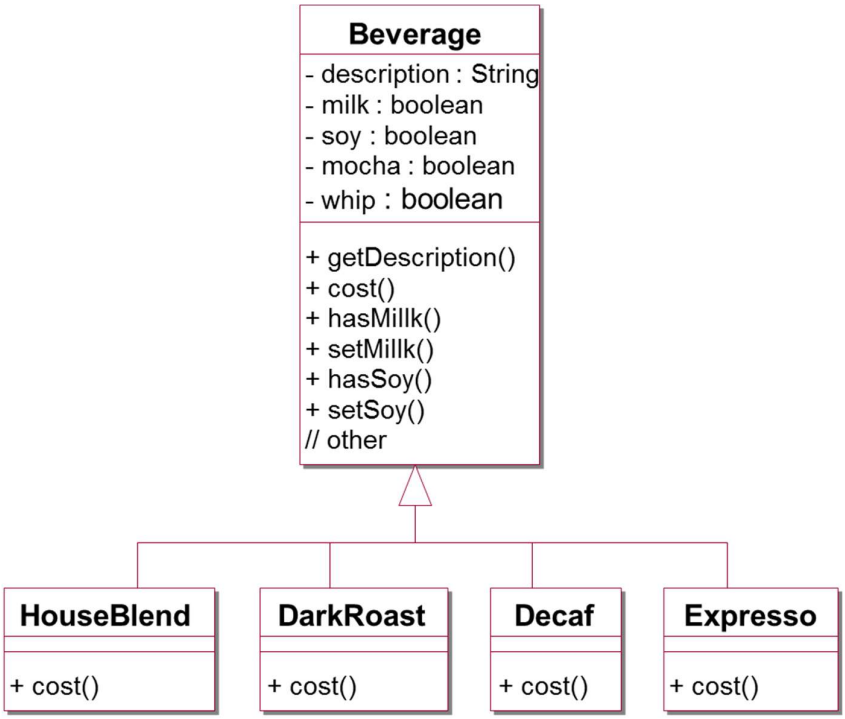
Các loại café và giá

- HouseBlend: 0.89 cent
- Decaf: 1.05 cent
- Espresso: 1.989 cent
- DarkRoast: 0.99 cent

Các loại gia vị

- Milk: 0.10 cent
- Soy: 0.15 cent
- Mocha: 0.20 cent
- Whip: 0.10 cent

Lab 1: Thiết kế lớp dùng kế thừa cho bài toán như sau

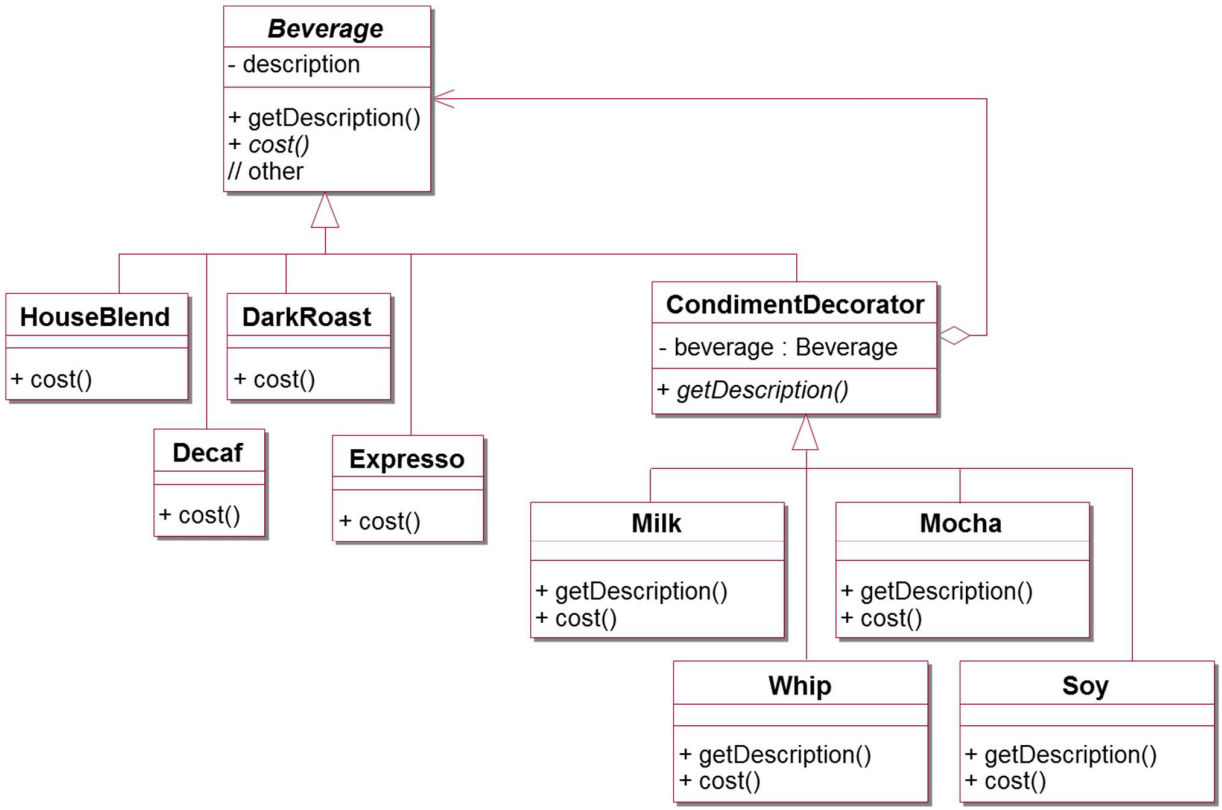


- Cài đặt phương thức **cost()** trong **Beverage** tính giá cho thành phần gia vị cộng thêm của café.
- Cài đặt phương thức **cost()** trong các lớp con để tính giá gồm giá café và giá thành phần gia vị cộng thêm của café.

Lab 2: Thêm các yêu cầu mới

- Thêm một loại gia vị mới
- Thêm gấp đôi một loại gia vị cho café.

Hiệu chỉnh thiết kế dùng mẫu Decorator



Cài đặt các lớp cho thiết kế và bổ sung các phần còn thiếu (// TODO)

```
public abstract class Beverage {
   String description = "Unknown Beverage";

   public String getDescription() {
      return description;
   }

   public abstract double cost();
}
```

```
public class HouseBlend extends Beverage {
   public HouseBlend() {
      description = "House Blend Coffee";
   }

   public double cost() {
      // TODO
   }
}
```

```
public class Decaf extends Beverage {
   public Decaf() {
      description = "Decaf Coffee";
   }

   public double cost() {
      // TODO
   }
}
```

```java
public class Espresso extends Beverage {
   public Espresso() {
      description = "Espresso";
   }

   public double cost() {
      // TODO
   }
}
```

```java
public class DarkRoast extends Beverage {
   public DarkRoast() {
      description = "Dark Roast Coffee";
   }

   public double cost() {
      // TODO
   }
}
```

```java
public abstract class CondimentDecorator extends Beverage {
   protected Beverage beverage;

   public CondimentDecorator(Beverage beverage) {
      this.beverage = beverage;
   }

   public abstract String getDescription();
}
```

```java
public class Milk extends CondimentDecorator {
   public Milk(Beverage beverage) {
      super(beverage);
   }

   public String getDescription() {
      // TODO
   }

   public double cost() {
      // TODO
   }
}
```

```java
public class Soy extends CondimentDecorator {
   public Soy(Beverage beverage) {
      super(beverage);
   }

   public String getDescription() {
      // TODO
   }

   public double cost() {
      // TODO
   }
}
```

```java
public class Mocha extends CondimentDecorator {
   public Mocha(Beverage beverage) {
      super(beverage);
   }

   public String getDescription() {
      // TODO
   }

   public double cost() {
      // TODO
   }
}
```

```java
public class Whip extends CondimentDecorator {
   public Whip(Beverage beverage) {
      super(beverage);
   }

   public String getDescription() {
      // TODO
   }

   public double cost() {
      // TODO
   }
}
```

```java
public class StarbuzzCoffee {
   public static void main(String args[]) {
      Beverage beverage = new Espresso();
      System.out.println(beverage.getDescription() + " $" + beverage.cost());

      Beverage beverage2 = new DarkRoast();
      beverage2 = new Mocha(beverage2);
      beverage2 = new Mocha(beverage2);
      beverage2 = new Whip(beverage2);
      System.out.println(beverage2.getDescription() + " $" + beverage2.cost());
      Beverage beverage3 = new HouseBlend();
      beverage3 = new Soy(beverage3);
      beverage3 = new Mocha(beverage3);
      beverage3 = new Whip(beverage3);
      System.out.println(beverage3.getDescription() + " $" + beverage3.cost());
   }
}
```

Bài 2: Decorate cho các nút lệnh của Swing như sau


```
public class Decorator extends JComponent {
   public Decorator(JComponent c) {
      setLayout(new BorderLayout());
      add("Center", c);
   }
}
```

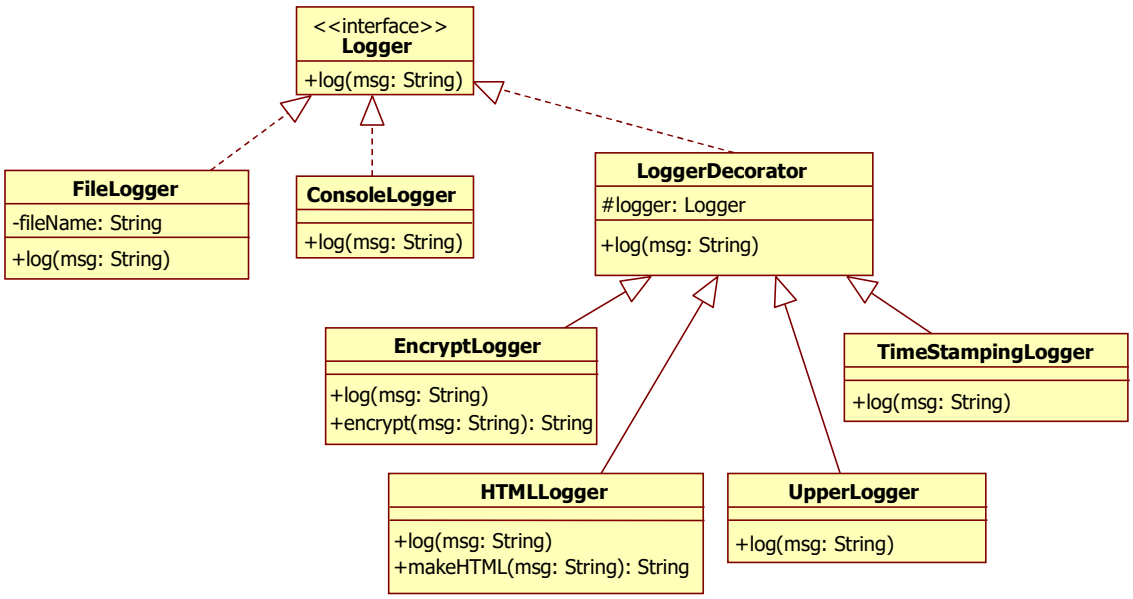
```
public class SlashDecorator extends Decorator {
   private int x1, y1, w1, h1;

   public SlashDecorator(JComponent c) {
      super(c);
   }

   public void setBounds(int x, int y, int w, int h) {
      x1 = x;
      y1 = y;
      w1 = w;
      h1 = h;
      super.setBounds(x, y, w, h);
   }

   public void paint(Graphics g) {
      super.paint(g);
      g.setColor(Color.red);
      g.drawLine(0, 0, w1, h1);
   }
}
```

```
public class CoolDecorator extends Decorator {
   private boolean mouse_over; // true when mose over button
   private JComponent thisComp;

   public CoolDecorator(JComponent c) {
      super(c);
      mouse_over = false;
      thisComp = this; // save this component
      // catch mouse movements in inner class
      c.addMouseListener(new MouseAdapter() {
         public void mouseEntered(MouseEvent e) {
            mouse_over = true; // set flag when mouse over
            thisComp.repaint();
         }

         public void mouseExited(MouseEvent e) {
            mouse_over = false; // clear flag when mouse not over
            thisComp.repaint();
         }
      });
   }
```

```
   // paint the button
   public void paint(Graphics g) {
      super.paint(g); // first draw the parent button
      if (!mouse_over) {
         // if the mouse is not over the button
         // erase the borders
         Dimension size = super.getSize();
         g.setColor(Color.lightGray);
         g.drawRect(0, 0, size.width - 1, size.height - 1);
         g.drawLine(size.width - 2, 0, size.width - 2, size.height - 1);
         g.drawLine(0, size.height - 2, size.width - 2, size.height - 2);
      }

   }
}
```

```
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class DecoWindow extends JFrame implements ActionListener {
   JButton quit;

   public DecoWindow() {
      super("Deco Button");
      addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e) {
            System.exit(0);
         }
      });

      JPanel jp = new JPanel();

      getContentPane().add(jp);
      jp.add(new CoolDecorator(new JButton("Cbutton")));
      jp.add(new SlashDecorator(new CoolDecorator(new JButton("Dbutton"))));
      // jp.add( new CoolDecorator(new JButton("Dbutton")));
      jp.add(quit = new JButton("Quit"));
      quit.addActionListener(this);
      setSize(new Dimension(200, 100));

      setVisible(true);
      quit.requestFocus();
   }

   public void actionPerformed(ActionEvent e) {
      System.exit(0);
   }

   static public void main(String argv[]) {
      new DecoWindow();
   }
}
```

Bài 3: Hệ thống ghi nhật ký Log. Các Logger có thể là ghi thông báo ra màn hình hoặc tập tin. Nội dung thông báo có thể được mã hóa, gắn thêm ngày giờ hiện thời, hoặc định dạng: đổi chữ in, chuyển sang mã HTML, … trước khi ghi nhật ký. Thiết kế các lớp Logger, có thể thêm các tính năng tùy ý như trên: mã hóa, gắn thêm ngày giờ hiện thời, hoặc định dạng.



Các Logger cơ bản

```
public interface Logger {
   public void log(String msg);
}

public class ConsoleLogger implements Logger {
   public void log(String msg) {
      System.out.println(msg);
   }
}
```

```
public class ConsoleLogger implements Logger {
   public void log(String msg) {
      System.out.println(msg);
   }
}
```

```
public class FileLogger implements Logger {
   private String fileName;

   public FileLogger(String fileName) {
      this.fileName = fileName;
   }

   public void log(String msg) {
      try {
         Writer out = new BufferedWriter(new OutputStreamWriter(
               new FileOutputStream(new File(fileName), true)));
         out.write(msg + "\n");
         out.close();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

LoggerDecorator

```
public class LoggerDecorator implements Logger {
   protected Logger logger;
   public LoggerDecorator(Logger logger) {
      this.logger = logger;
   }

   public void log(String msg) {
      // Default implementation to be overriden by subclasses.
      logger.log(msg);
   }

}
```

```
public class HTMLLogger extends LoggerDecorator {

   public HTMLLogger(Logger logger) {
      super(logger);
   }

   public void log(String msg) {
      // Added functionality
      msg = makeHTML(msg);

      // Now forward the encrypted text to the FileLogger for storage
      logger.log(msg);
   }

   public String makeHTML(String msg) {
      // Make it into an HTML document.
      msg = "<HTML><BODY>" + "<b>" + msg + "</b>" + "</BODY></HTML>";
      return msg;
   }

}
```

```
public class EncryptLogger extends LoggerDecorator {

   public EncryptLogger(Logger logger) {
      super(logger);
   }

   public void log(String msg) {
      // Added functionality
      msg = encrypt(msg);

      // Now forward the encrypted text to the FileLogger for storage
      logger.log(msg);
   }

   public String encrypt(String msg) {
      // Apply simple encryption by Transposition...
      // Shift all characters by one position.
      msg = msg.substring(msg.length() - 1)
            + msg.substring(0, msg.length() - 1);
      return msg;
   }
}
```

```
public class TimeStampingLogger extends LoggerDecorator {

   public TimeStampingLogger(Logger logger) {
      super(logger);
   }

   public void log(String msg) {
      Date now = new Date();
      msg = now + "" + msg;
      logger.log(msg);
   }
}
```

```
public class UpperLogger extends LoggerDecorator {
    public UpperLogger(Logger logger) {
        super(logger);
    }

    public void log(String msg) {
        logger.log(msg.toUpperCase());
    }
}
```

```
public class LoggerFactory {
   public boolean isFileLoggingEnabled() {
      Properties p = new Properties();
      try {
         p.load(ClassLoader.getSystemResourceAsStream("Logger.properties"));
         String fileLoggingValue = p.getProperty("FileLogging");
         if (fileLoggingValue.equalsIgnoreCase("ON") == true)
            return true;
         else
            return false;
      } catch (IOException e) {
         return false;
      }
   }

   public Logger getLogger() {
      if (isFileLoggingEnabled()) {
         return new FileLogger("log.txt");
      } else {
         return new ConsoleLogger();
      }
   }
}
```

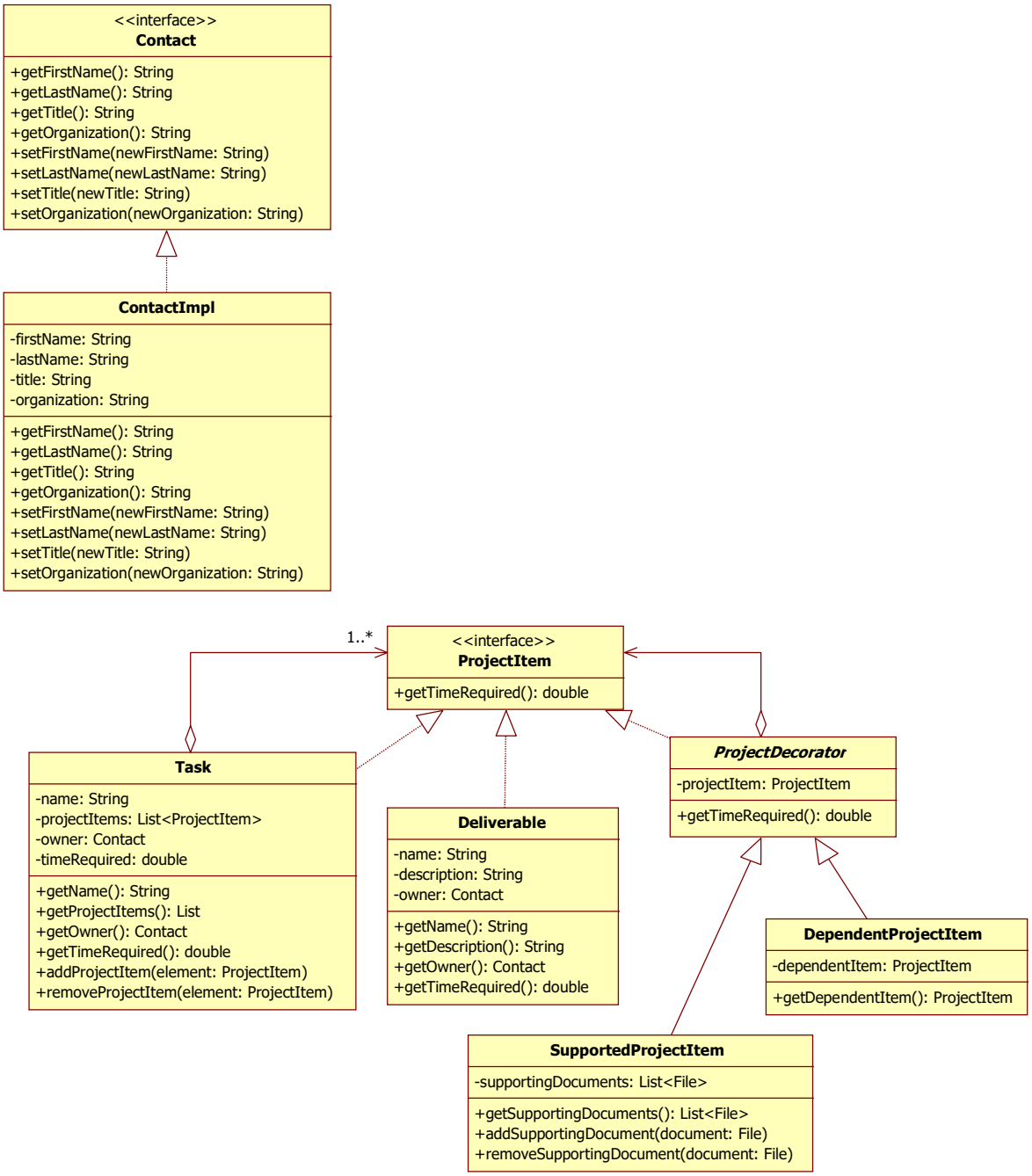
```
public class DecoratorClient {
   public static void main(String[] args) {
      LoggerFactory factory = new LoggerFactory();
      Logger logger = factory.getLogger();

      HTMLLogger hLogger = new HTMLLogger(logger);
```

```
        // the decorator object provides the same interface.
        hLogger.log("A Message to Log");

        EncryptLogger eLogger = new EncryptLogger(logger);
        eLogger.log("A Message to Log");

        Logger uLogger = new UpperLogger(new TimeStampingLogger(logger));
        uLogger.log("A Message to Log");
    }
}
```

File **Logger.properties**

```
FileLogging=OFF
```

Bài 4: Hệ thống quản lý dự án
Mỗi dự án có thể là




```
public interface Contact extends Serializable {
   public static final String SPACE = " ";
   public String getFirstName();
   public String getLastName();
   public String getTitle();
   public String getOrganization();
   public void setFirstName(String newFirstName);
   public void setLastName(String newLastName);
   public void setTitle(String newTitle);
   public void setOrganization(String newOrganization);
}
```

```
public class ContactImpl implements Contact {
   private String firstName;
   private String lastName;
   private String title;
   private String organization;

   public ContactImpl() { }

   public ContactImpl(String newFirstName, String newLastName, String newTitle,
         String newOrganization) {
      firstName = newFirstName;
      lastName = newLastName;
      title = newTitle;
      organization = newOrganization;
   }

   public String getFirstName() {
      return firstName;
   }

   public String getLastName() {
      return lastName;
   }

   public String getTitle() {
      return title;
   }

   public String getOrganization() {
      return organization;
   }

   public void setFirstName(String newFirstName) {
      firstName = newFirstName;
   }

   public void setLastName(String newLastName) {
      lastName = newLastName;
   }

   public void setTitle(String newTitle) {
      title = newTitle;
   }

   public void setOrganization(String newOrganization) {
      organization = newOrganization;
   }

   public String toString() {
      return firstName + SPACE + lastName;
   }
}
```

```
public interface ProjectItem extends Serializable {
   public static final String EOL_STRING = System.getProperty("line.separator");

   public double getTimeRequired();
}
```

```
public class Task implements ProjectItem {
   private String name;
   private List<ProjectItem> projectItems = new ArrayList<ProjectItem>();
   private Contact owner;
   private double timeRequired;

   public Task() { }

   public Task(String newName, Contact newOwner, double newTimeRequired) {
      name = newName;
      owner = newOwner;
      timeRequired = newTimeRequired;
   }

   public String getName() {
      return name;
   }

   public List<ProjectItem> getProjectItems() {
      return projectItems;
   }
   public Contact getOwner() {
      return owner;
   }

   public double getTimeRequired() {
      double totalTime = timeRequired;
      Iterator<ProjectItem> items = projectItems.iterator();
      while (items.hasNext()) {
         ProjectItem item = items.next();
         totalTime += item.getTimeRequired();
      }
      return totalTime;
   }

   public void setName(String newName) {
      name = newName;
   }

   public void setOwner(Contact newOwner) {
      owner = newOwner;
   }

   public void setTimeRequired(double newTimeRequired) {
      timeRequired = newTimeRequired;
   }

   public void addProjectItem(ProjectItem element) {
      if (!projectItems.contains(element)) {
         projectItems.add(element);
      }
   }

   public void removeProjectItem(ProjectItem element) {
      projectItems.remove(element);
   }

   public String toString() {
      return "Task: " + name;
   }
```

```
}
```

```
public class Deliverable implements ProjectItem {
   private String name;
   private String description;
   private Contact owner;

   public Deliverable() { }

   public Deliverable(String newName, String newDescription, Contact newOwner) {
      name = newName;
      description = newDescription;
      owner = newOwner;
   }

   public String getName() {
      return name;
   }

   public String getDescription() {
      return description;
   }

   public Contact getOwner() {
      return owner;
   }

   public double getTimeRequired() {
      return 0;
   }

   public void setName(String newName) {
      name = newName;
   }

   public void setDescription(String newDescription) {
      description = newDescription;
   }

   public void setOwner(Contact newOwner) {
      owner = newOwner;
   }

   public String toString() {
      return "Deliverable: " + name;
   }
}
```

```
public abstract class ProjectDecorator implements ProjectItem {
   private ProjectItem projectItem;

   protected ProjectItem getProjectItem() {
      return projectItem;
   }

   public void setProjectItem(ProjectItem newProjectItem) {
      projectItem = newProjectItem;
   }

   // Default implementation to be overriden by subclasses.
   public double getTimeRequired() {
      return projectItem.getTimeRequired();
   }
}
```

```
public class DependentProjectItem extends ProjectDecorator {
   private ProjectItem dependentItem;

   public DependentProjectItem() { }

   public DependentProjectItem(ProjectItem newDependentItem) {
      dependentItem = newDependentItem;
   }

   public ProjectItem getDependentItem() {
      return dependentItem;
   }

   public void setDependentItem(ProjectItem newDependentItem) {
      dependentItem = newDependentItem;
   }

   public String toString() {
      return getProjectItem().toString() + EOL_STRING
            + "\tProjectItem dependent on: " + dependentItem;
   }
}
```

```
public class SupportedProjectItem extends ProjectDecorator {
   private List<File> supportingDocuments = new ArrayList<File>();

   public SupportedProjectItem() { }
   public SupportedProjectItem(File newSupportingDocument) {
      addSupportingDocument(newSupportingDocument);
   }

   public List<File> getSupportingDocuments() {
      return supportingDocuments;
   }

   public void addSupportingDocument(File document) {
      if (!supportingDocuments.contains(document)) {
         supportingDocuments.add(document);
      }
   }
   public void removeSupportingDocument(File document) {
      supportingDocuments.remove(document);
   }
```

```
   public String toString() {
      return getProjectItem().toString() + EOL_STRING
            + "\tSupporting Documents: " + supportingDocuments;
   }
}
```

```
public class RunDecoratorPattern {
   public static void main(String[] arguments) {
      System.out.println("Example for the Decorator pattern");
      System.out.println();
      System.out.println("This demonstration will show how Decorator classes can"
         + " be used to extend the basic functionality of ProjectItems."
         + " The Task and Deliverable classes provide the basic ProjectItems,"
         + " and their functionality will be extended by adding subclasses of the"
         + " abstract class ProjectDecorator.");
      System.out.println();
      System.out.println("Note that the toString method has been overridden for all"
         + " ProjectItems, to more effectively show how Decorators are associated with"
         + " their ProjectItems.");
      System.out.println();

      System.out.println("Creating ProjectItems.");
      Contact contact1 = new ContactImpl("Simone", "Roberto",
        "Head Researcher and Chief Archivist", "Institute for Advanced (Java) Studies");
      Task task1 = new Task("Perform months of diligent research", contact1, 20.0);
      Task task2 = new Task("Obtain grant from World Java Foundation", contact1, 40.0);
      Deliverable deliverable1 = new Deliverable("Java History",
            "Comprehensive history of the design of all Java APIs", contact1);
      System.out.println("ProjectItem objects created. Results:");
      System.out.println(task1);
      System.out.println(task2);
      System.out.println(deliverable1);
      System.out.println();

      System.out.println("Creating decorators");
      ProjectDecorator decorator1 =
           new SupportedProjectItem(new File("JavaHistory.txt"));
      ProjectDecorator decorator2 = new DependentProjectItem(task2);
      System.out.println("Decorators created. Adding decorators to the first task");
      decorator1.setProjectItem(task1);
      decorator2.setProjectItem(decorator1);
      System.out.println();
      System.out.println("Decorators added. Results");
      System.out.println(decorator2);
   }
}
```