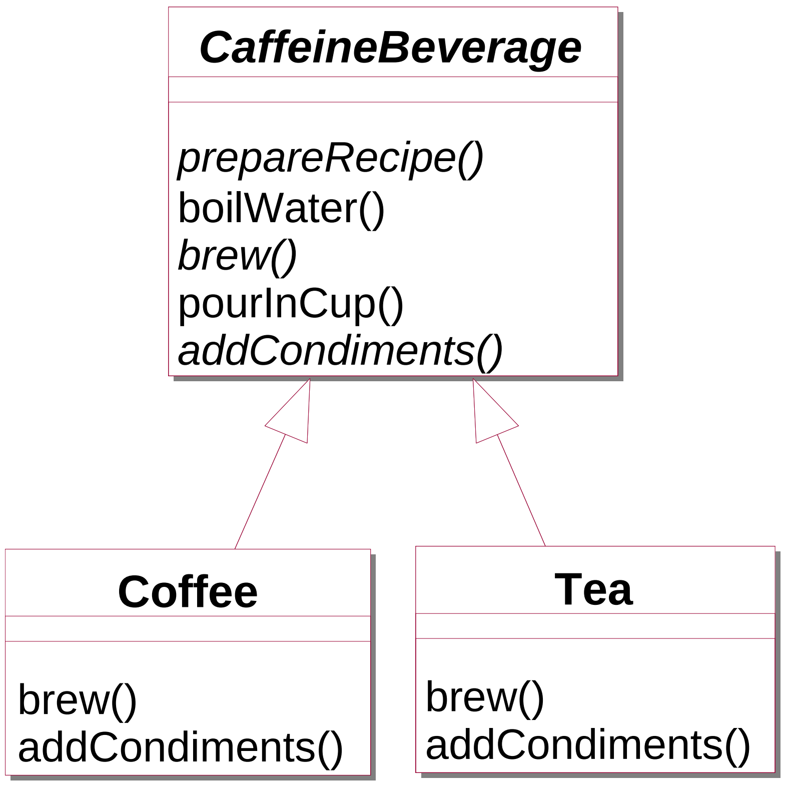
Template method Pattern

Lab 0:



```java
public abstract class CaffeineBeverage {

   final void prepareRecipe() {
      boilWater();
      brew();
      pourInCup();
      addCondiments();
   }

   abstract void brew();

   abstract void addCondiments();

   void boilWater() {
      System.out.println("Boiling water");
   }

   void pourInCup() {
      System.out.println("Pouring into cup");
   }
}
```

```java
public class Coffee extends CaffeineBeverage {
   public void brew() {
      System.out.println("Dripping Coffee through filter");
   }
   public void addCondiments() {
      System.out.println("Adding Sugar and Milk");
   }
}
```

```java
public class Tea extends CaffeineBeverage {
   public void brew() {
      System.out.println("Steeping the tea");
   }

   public void addCondiments() {
      System.out.println("Adding Lemon");
   }
}
```

```java
public abstract class CaffeineBeverageWithHook {

   void prepareRecipe() {
      boilWater();
      brew();
      pourInCup();
      if (customerWantsCondiments()) {
         addCondiments();
      }
   }

   abstract void brew();

   abstract void addCondiments();

   void boilWater() {
      System.out.println("Boiling water");
   }

   void pourInCup() {
      System.out.println("Pouring into cup");
   }

   boolean customerWantsCondiments() {
      return true;
   }
}
```

```java
public class CoffeeWithHook extends CaffeineBeverageWithHook {

   public void brew() {
      System.out.println("Dripping Coffee through filter");
   }

   public void addCondiments() {
      System.out.println("Adding Sugar and Milk");
   }

   public boolean customerWantsCondiments() {

      String answer = getUserInput();

      if (answer.toLowerCase().startsWith("y")) {
         return true;
      } else {
         return false;
      }
   }

   private String getUserInput() {
      String answer = null;

      System.out
            .print("Would you like milk and sugar with your coffee (y/n)? ");

      BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
      try {
         answer = in.readLine();
      } catch (IOException ioe) {
         System.err.println("IO error trying to read your answer");
      }
      if (answer == null) {
         return "no";
      }
      return answer;
   }
}
```

```java
public class TeaWithHook extends CaffeineBeverageWithHook {

   public void brew() {
      System.out.println("Steeping the tea");
   }

   public void addCondiments() {
      System.out.println("Adding Lemon");
   }

   public boolean customerWantsCondiments() {

      String answer = getUserInput();

      if (answer.toLowerCase().startsWith("y")) {
         return true;
      } else {
         return false;
      }
   }

   private String getUserInput() {
      // get the user's response
      String answer = null;

      System.out
            .print("Would you like milk and sugar with your coffee (y/n)? ");

      BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
      try {
         answer = in.readLine();
      } catch (IOException ioe) {
         System.err.println("IO error trying to read your answer");
      }
      if (answer == null) {
         return "no";
      }
      return answer;
   }
}
```

```java
public class BeverageTestDrive {
   public static void main(String[] args) {

      Tea tea = new Tea();
      Coffee coffee = new Coffee();

      System.out.println("\nMaking tea...");
      tea.prepareRecipe();

      System.out.println("\nMaking coffee...");
      coffee.prepareRecipe();

      TeaWithHook teaHook = new TeaWithHook();
      CoffeeWithHook coffeeHook = new CoffeeWithHook();

      System.out.println("\nMaking tea...");
      teaHook.prepareRecipe();

      System.out.println("\nMaking coffee...");
      coffeeHook.prepareRecipe();
   }
}
```

Lab 1:

We wish to write a ***String Decorator*** class, which decorates the given string by appending some characters at the beginning, left, right and top of the string.

For that, let us write a class which looks like the following,

StringDecorator.java

```java
public abstract class StringDecorator {

   // template method
   public final void decorate() {
      char topChar = getTopCharacter();
      char leftChar = getLeftCharacter();
      String str = getString();
      char rightChar = getRightCharacter();
      char bottomChar = getBottomCharacter();

      for (int i = 0; i < str.length() + 2; i++) {
         System.out.print(topChar);
      }
      System.out.println();

      System.out.print(leftChar);
      if (isUpperCase()) {
         System.out.print(str.toUpperCase());
      } else {
         System.out.print(str.toLowerCase());
      }

      System.out.print(rightChar);

      System.out.println();
      for (int i = 0; i < str.length() + 2; i++) {
         System.out.print(bottomChar);
      }
   }

   // hook method
   protected boolean isUpperCase() {
      return true;
   }

   protected abstract char getTopCharacter();

   protected abstract char getLeftCharacter();

   protected abstract String getString();

   protected abstract char getRightCharacter();

   protected abstract char getBottomCharacter();
}
```

The above `decorate()` method contains the original logic for decorating a string object, which in turns calls the abstract methods to get the various characters for decorating. Since this method executes the core logic flow, this method is often called as the ***template method***.

Now, let us see one concrete implementation of the above class. Given below is the code snippet for the same,

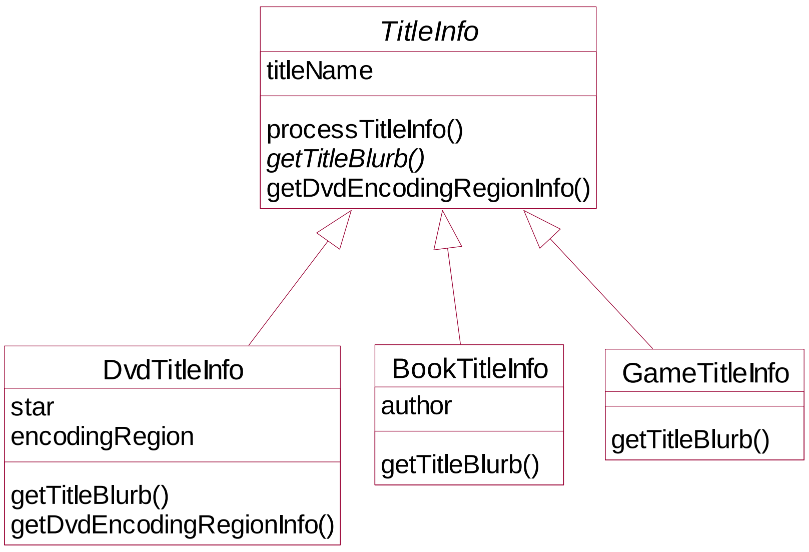
SimpleStringDecorator.java

```
public class SimpleStringDecorator extends StringDecorator {

   protected char getBottomCharacter() {
      return '#';
   }

   protected char getLeftCharacter() {
      return '(';
   }

   protected char getRightCharacter() {
      return ')';
   }

   protected String getString() {
      return "javabeat.net";
   }
   protected char getTopCharacter() {
      return '#';
   }

   public static void main(String[] args) {
      StringDecorator decorator = new SimpleStringDecorator();
      decorator.decorate();
   }
}
```

See how the above class redefines the abstract methods thereby providing its own set of characters for decorating the string object. If you run the above program, you will see in the console similar to the following,

```
##############
(JAVABEAT.NET)
##############
```

Lab 2:



```
public abstract class TitleInfo {
   private String titleName;

   // the "template method" -
   // calls the concrete class methods, is not overridden
   public final String processTitleInfo() {
      StringBuffer titleInfo = new StringBuffer();

      titleInfo.append(this.getTitleBlurb());
      titleInfo.append(this.getDvdEncodingRegionInfo());

      return titleInfo.toString();
   }

   // the following 2 methods are "concrete abstract class methods"
   public final void setTitleName(String titleNameIn) {
```

```java
      this.titleName = titleNameIn;
   }

   public final String getTitleName() {
      return this.titleName;
   }

   // this is a "primitive operation",
   // and must be overridden in the concrete templates
   public abstract String getTitleBlurb();

   // this is a "hook operation", which may be overridden,
   // hook operations usually do nothing if not overridden
   public String getDvdEncodingRegionInfo() {
      return " ";
   }
}
```

```java
class DvdTitleInfo extends TitleInfo {
   String star;
   char encodingRegion;

   public DvdTitleInfo(String titleName, String star, char encodingRegion) {
      this.setTitleName(titleName);
      this.setStar(star);
      this.setEncodingRegion(encodingRegion);
   }

   public char getEncodingRegion() {
      return encodingRegion;
   }

   public void setEncodingRegion(char encodingRegion) {
      this.encodingRegion = encodingRegion;
   }

   public String getStar() {
      return star;
   }

   public void setStar(String star) {
      this.star = star;
   }

   public String getTitleBlurb() {
      return ("DVD: " + this.getTitleName() + ", starring " + this.getStar());
   }

   public String getDvdEncodingRegionInfo() {
      return (", encoding region: " + this.getEncodingRegion());
   }
}
```

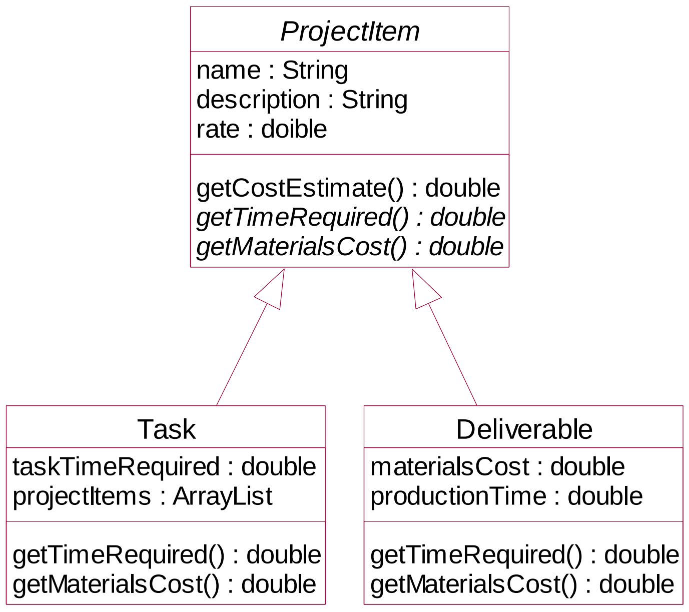
```java
class BookTitleInfo extends TitleInfo {
   private String author;

   public BookTitleInfo(String titleName, String author) {
      this.setAuthor(author);
```

```java
        this.setTitleName(titleName);
    }

    public String getAuthor() {
        return author;
    }

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

    public String getTitleBlurb() {
        return ("Book: " + this.getTitleName() + ", Author: " + this.getAuthor());
    }
}
```

```java
public class GameTitleInfo extends TitleInfo {
    public GameTitleInfo(String titleName) {
        this.setTitleName(titleName);
    }

    public String getTitleBlurb() {
        return ("Game: " + this.getTitleName());
    }
}
```

```java
public class TemplatePatternDemo {

    public static void main(String[] args) {
        TitleInfo bladeRunner = new DvdTitleInfo("Blade Runner", "Harrison Ford",
                '1');
        TitleInfo electricSheep = new BookTitleInfo(
                "Do Androids Dream of Electric Sheep?", "Philip");

        System.out.println(" ");
        System.out
                .println("Testing bladeRunner" + bladeRunner.processTitleInfo());
        System.out.println("Testing electricSheep"
                + electricSheep.processTitleInfo());
    }
}
```

Lab 3:



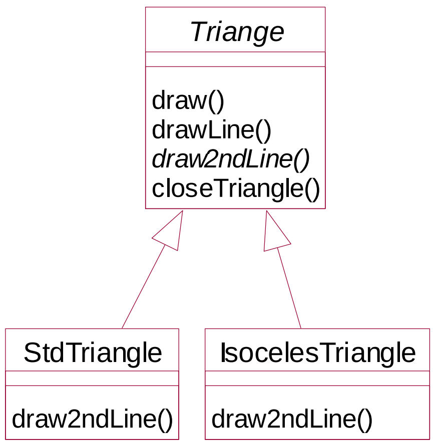
ProjectItem
name : String
description : String
rate : doible
getCostEstimate() : double
getTimeRequired() : double
getMaterialsCost() : double
Task
taskTimeRequired : double
projectItems : ArrayList
getTimeRequired() : double
getMaterialsCost() : double
Deliverable
materialsCost : double
productionTime : double
getTimeRequired() : double
getMaterialsCost() : double

```java
abstract class ProjectItem implements Serializable {
   private String name;
   private String description;
   private double rate;

   public ProjectItem() {
   }

   public ProjectItem(String newName, String newDescription, double newRate) {
      name = newName;
      description = newDescription;
      rate = newRate;
   }

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

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

   public void setRate(double newRate) {
      rate = newRate;
   }

   public String getName() {
      return name;
   }

   public String getDescription() {
      return description;
   }

   // template method
   public final double getCostEstimate() {
      return getTimeRequired() * getRate() + getMaterialsCost();
   }
   public double getRate() {
      return rate;
   }

   public String toString() {
      return getName();
   }

   public abstract double getTimeRequired();

   public abstract double getMaterialsCost();
}

class Task extends ProjectItem {
   private ArrayList projectItems = new ArrayList();
   private double taskTimeRequired;

   public Task() {
   }

   public Task(String newName, String newDescription,
         double newTaskTimeRequired, double newRate) {
      super(newName, newDescription, newRate);
      taskTimeRequired = newTaskTimeRequired;
   }

   public void setTaskTimeRequired(double newTaskTimeRequired) {
      taskTimeRequired = newTaskTimeRequired;
   }

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

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

   public double getTaskTimeRequired() {
      return taskTimeRequired;
   }

   public Iterator getProjectItemIterator() {
      return projectItems.iterator();
   }

   public double getMaterialsCost() {
      double totalCost = 0;
      Iterator items = getProjectItemIterator();
      while (items.hasNext()) {
         totalCost += ((ProjectItem) items.next()).getMaterialsCost();
      }
      return totalCost;
   }

   public double getTimeRequired() {
      double totalTime = taskTimeRequired;
      Iterator items = getProjectItemIterator();
      while (items.hasNext()) {
         totalTime += ((ProjectItem) items.next()).getTimeRequired();
      }
      return totalTime;
   }
}
```

```java
class Deliverable extends ProjectItem {
   private double materialsCost;
   private double productionTime;

   public Deliverable() {
   }

   public Deliverable(String newName, String newDescription,
         double newMaterialsCost, double newProductionTime, double newRate) {
      super(newName, newDescription, newRate);
      materialsCost = newMaterialsCost;
      productionTime = newProductionTime;
   }

   public void setMaterialsCost(double newCost) {
      materialsCost = newCost;
   }

   public void setProductionTime(double newTime) {
      productionTime = newTime;
   }

   public double getMaterialsCost() {
      return materialsCost;
   }

   public double getTimeRequired() {
      return productionTime;
   }
}
```

```java
public class RunTemplateMethodPattern {
   public static void main(String[] arguments) {
      System.out.println("Example for the Template Method pattern");
      System.out.println("This code demonstrates how the template method can");
      System.out.println(" be used to define a variable implementation for a");
      System.out.println(" common operation. In this case, the ProjectItem");
      System.out.println(" abstract class defines the method getCostEstimate,");
      System.out.println(" which is a combination of the cost for time and");
      System.out.println(" materials. The two concrete subclasses used here,");
      System.out.println(" Task and Deliverable, have different methods of");
      System.out.println(" providing a cost estimate.");
      System.out.println();

      System.out.println("Creating a demo Task and Deliverable");
      System.out.println();
      Task primaryTask = new Task("Put a JVM on the moon",
            "Lunar mission as part of the JavaSpace program ;)", 240.0, 100.0);
      primaryTask.addProjectItem(new Task("Establish ground control", "",
            1000.0, 10.0));
      primaryTask.addProjectItem(new Task("Train the Javanaughts", "", 80.0,
            30.0));
      Deliverable deliverableOne = new Deliverable(
            "Lunar landing module",
            "Ask the local garage if they can make a few minor modifications to one of
their cars",
            2800, 40.0, 35.0);

      System.out
            .println("Calculating the cost estimates using the Template Method,
getCostEstimate.");
      System.out.println();
      System.out.println("Total cost estimate for: " + primaryTask);
      System.out.println("\t" + primaryTask.getCostEstimate());
      System.out.println();

      System.out.println("Total cost estimate for: " + deliverableOne);
      System.out.println("\t" + deliverableOne.getCostEstimate());
   }
}
```

Lab 4:



```
abstract class Triangle {
   Point p1, p2, p3;

   public Triangle(Point a, Point b, Point c) {
      // save
      p1 = a;
      p2 = b;
      p3 = c;
   }

   // template method
   public void draw(Graphics g) {
      // This routine draws a general triangle
      drawLine(g, p1, p2);
      Point current = draw2ndLine(g, p2, p3);
      closeTriangle(g, current);
   }

   public void drawLine(Graphics g, Point a, Point b) {
      g.drawLine(a.x, a.y, b.x, b.y);
```

```java
   }

   // this routine is the "Hook" that has to be implemented
   // for each triangle type.
   abstract public Point draw2ndLine(Graphics g, Point a, Point b);

   public void closeTriangle(Graphics g, Point c) {
      // draw back to first point
      g.drawLine(c.x, c.y, p1.x, p1.y);
   }

}
```

```java
class StdTriangle extends Triangle {
   public stdTriangle(Point a, Point b, Point c) {
      super(a, b, c);
   }

   public Point draw2ndLine(Graphics g, Point a, Point b) {
      g.drawLine(a.x, a.y, b.x, b.y);
      return b;
   }
}
```

```java
class IsoscelesTriangle extends Triangle {
   Point newc;

   int newcx, newcy;

   int incr;

   public IsoscelesTriangle(Point a, Point b, Point c) {
      super(a, b, c);
      double dx1 = b.x - a.x;
      double dy1 = b.y - a.y;
      double dx2 = c.x - b.x;
      double dy2 = c.y - b.y;

      double side1 = calcSide(dx1, dy1);
      double side2 = calcSide(dx2, dy2);

      if (side2 < side1)
         incr = -1;
      else
         incr = 1;

      double slope = dy2 / dx2;
      double intercept = c.y - slope * c.x;

      // move point c so that this is an isoceles triangle
      newcx = c.x;
      newcy = c.y;
      while (Math.abs(side1 - side2) > 1) {
         newcx += incr; // iterate a pixel at a time until close
         newcy = (int) (slope * newcx + intercept);
         dx2 = newcx - b.x;
         dy2 = newcy - b.y;
         side2 = calcSide(dx2, dy2);
      }
      newc = new Point(newcx, newcy);
   }

   // calculate length of side
   private double calcSide(double dx, double dy) {
      return Math.sqrt(dx * dx + dy * dy);
   }

   // draws 2nd line using saved new point
   public Point draw2ndLine(Graphics g, Point b, Point c) {
      g.drawLine(b.x, b.y, newc.x, newc.y);
      return newc;
   }
}
```

```
public class TriangleDrawing extends JFrame {
   stdTriangle t, t1;

   IsocelesTriangle it;

   public TriangleDrawing() {
      super("Draw triangles");
      addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e) {
            System.exit(0);
         }
      });

      TPanel tp = new TPanel();
      t = new StdTriangle(new Point(10, 10), new Point(150, 50), new Point(100,
            75));
      it = new IsocelesTriangle(new Point(150, 100), new Point(240, 40),
            new Point(175, 150));
      t1 = new StdTriangle(new Point(150, 100), new Point(240, 40), new Point(
            175, 150));
      tp.addTriangle(t);
      tp.addTriangle(it);
      // tp.addTriangle(t1);

      getContentPane().add(tp);
      setSize(300, 200);
      setBackground(Color.white);
      setVisible(true);
   }

   public static void main(String[] arg) {
      new TriangleDrawing();
   }
}
```

```
class TPanel extends JPanel {
   Vector triangles;

   public TPanel() {
      triangles = new Vector();
   }
   public void addTriangle(Triangle t) {
      triangles.addElement(t);
   }

   public void paint(Graphics g) {
      for (int i = 0; i < triangles.size(); i++) {
         Triangle tngl = (Triangle) triangles.elementAt(i);
         tngl.draw(g);
      }
   }
}
```