Requirements:

R.1 Implement a program written in the Java language to instantiate and examine objects of the PC component class(es) listed in the Problem Statement (appended).

R.2 Include as a resource a database that contains these properties of the PCComponent classes:

R.2.1 component name (explicit requirement)

R.2.2 component self cost (explicit requirement)

R.2.3 subsidiary component list (inferred requirement)

R.3 Implement a user interface through which the user can select a component for examination (explicit req.) and view its properties (implicit).

R.3.1 To make the selection meaningful, there shall be more than one component presented in the list (implicit) from which the user selects.

R.3.2 Information shall be presented to the user in a way that is well-formatted (explicit req.), arranged to simplify visual recognition of which properties belong to which component (implicit).

Analysis:

A.1 Resources:

Staff: one software engineer

Time: about one week, starting 30 September 2004. (Most of the preceding week was wasted trying to find consensus with other team members.) NOTE: By 30 September the database tables had already been defined and implemented in MySQL.

Tools: MySQL, type 4 JDBC driver. NetBeans IDE.

A.2 Work Priorities:

A.2.1 Design and debug a PCComponent class to model the components listed in the Problem Statement. Because the components learn their properties from the database (Requirement 2), the database must be ready before PCComponent testing can begin.

A.2.1.1 Design the database first and activate it. This requires that database engine MySQL be installed, JDBC driver for MySQL be present in JDBCHOME, operating system variables JAVA_HOME and JDBCHOME be defined.

A.2.1.2 Design the PCComponent class.

A.2.1.3 Code a unit test driver for the PCComponent class.

A.2.2 Implement a style of user interface that satisfies the requirements with the least amount of design and debug time.

A.2.3 Collect and polish the (paperless) paperwork.

Design:

D.1 Database design:

D.1.1 For each type of PC component enumerated in the Problem Statement there shall be a *component table*.

D.1.2 Each row of a component table shall describe a particular instance of that table's component type.

D.1.3 The columns of a component table shall include

D.1.3.1 A unique identifier per row (prikey)

D.1.3.2 A component instance name (name)

D.1.3.3 A component instance cost (owncost)

D.1.4 If the kind of component a component table describes includes subcomponents, there shall be columns in the table that name each type of subcomponent. The column name shall

D.1.5 There shall be one component naming table that associates the names of component tables with descriptive names for the kinds of components they define. For example, the descriptive name could be "External Speakers" for the component table **speakers**.

D.2 Component Class design

Deviation from the Problem Statement: Literal adherence to the problem statement would require creation of seventeen distinct classes. This solution shall define one base class (PCComponent) that when instantiated becomes, depending on the parameters passed to its constructor, an object that models any one of the component types enumerated in the Problem Statement.

D.2.1 A PCComponent instance shall have a **cost()** method that returns the cost of the instance (explicit, from the Problem Statement).

D.2.1.1 Because a PCComponent may include subcomponents, there must be a way for an instance's cost() method to return the sum of its own cost and the costs of its subcomponents.

D.2.1.2 So that an instance can readily find its subcomponents, instantiation of a class shall instantiate all its subcomponents. The class' properties shall include references to all the subcomponents.

D.2.2 A component's toString() method shall return a 'a nicely formatted string containing a recursive listing of the components making up a component.' (from the Problem Statement)

D.2.2.1 The *toString()*s from subcomponents shall be appended to the toString() of the parent component. Each subcomponent's toString() shall be appended on a new line, indented by from the start of the parent component's description string by one tab ('\t') .

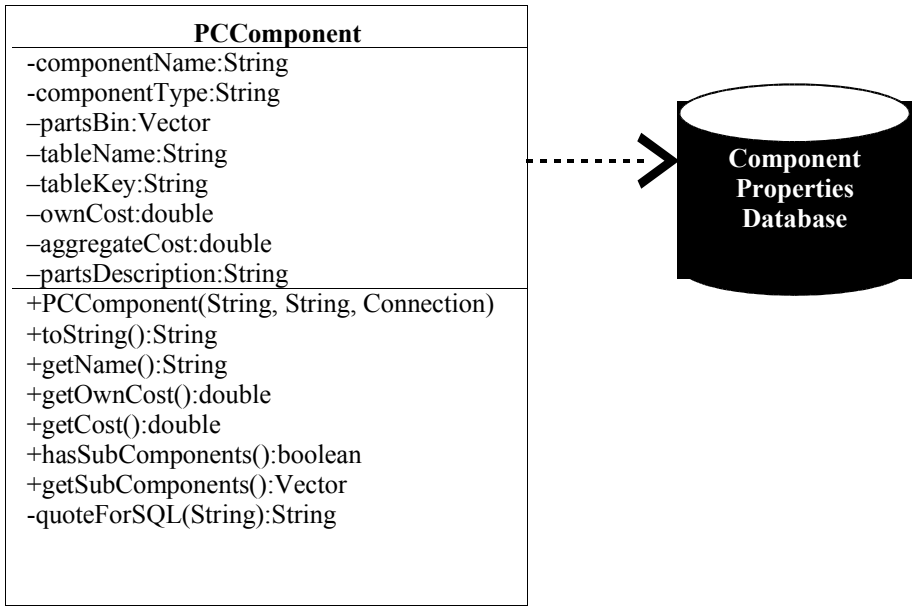
Example:

parent_type instance_name own_cost

<tab>child_type instance_name own_cost

<tab>child_type instance_name own_cost

<tab><tab>grandchild_type instance_name own_cost



D.3 User Interface class design

D.3.1 The user interface class(es) shall implement a text mode (console) UI, because that is the only kind that I can hope to complete in the time allowed. (A.2.2)

D.3.2 The UI shall be organized as screen pages.

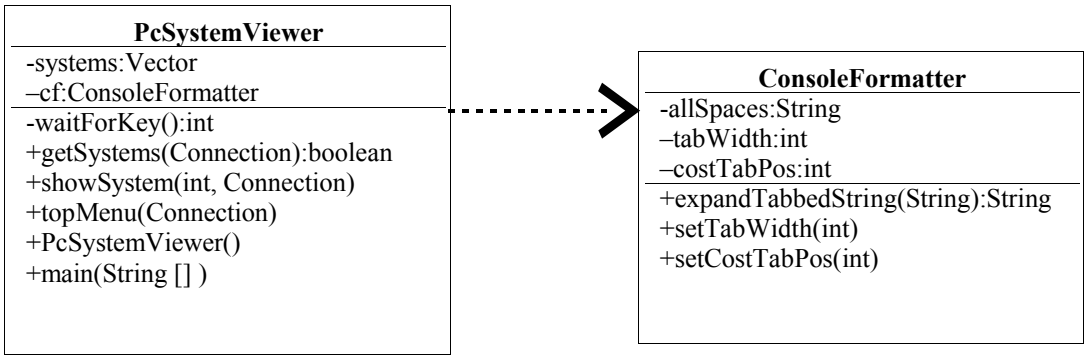
D.3.2.1 The page presented to the user at program start shall show a list of components by name and number. By selecting a number, the user may invoke a detail page (D.3.2.2) that shall show the properties of the selected component and all its subcomponents.

The program start page shall also provide a way for the user to terminate the program.

D.3.2.2 The detail page shall show the properties of the selected component by printing the selected component's toString().

After printing the toString(), the detail page shall wait for a keypress from the user before returning to the component list page (D.3.2.1)

(Classes derived from interface design specification)



Coding:

Design and code the classes in this order:

FiatData.java	Populates the 'task2' database. It assumes 'task2' has been created and is known to the SQL engine.
PCComponent.java	Implements the generic component class. The constructor gets the name of a database table, the key for a row in that table, and a Connection to the database. The constructor then builds the instance per the data retrieved from the database, calling subcomponent constructors recursively if needed.
PCComponentTest.java	A bare-bones driver for verifying PCComponent functionality.
ConsoleFormatter.java	A helper class that massages the raw toString() output from PCComponent, changing the tabbing size and lining up the cost() numbers at the right screen margin.
PcSystemViewer.java	The console-mode UI.

See the **javadoc** folder for more information on using these classes.

Testing:

Unit test for PCComponent class

Start the database.

In the IDE, load **PCComponentTest.java** into the editor. Uncomment line 32, and comment out lines 33 and 34. Run the program. The IDE's Output Window should show

```
Got data for Processor: PIII 866 Coppermine39.95
  system cost = $39.95
```

This verifies correct construction of a PCComponent instance of the simplest form, one with no subcomponents. It also verifies the simplest case operations of the toString() and getCost() methods.

Comment out line 32, and uncomment line 33. Run the program. The IDE's Output Window should show

```
Got data for System main board: Business MB136.99
      Processor: Athlon XP 240074.99
      On-board Memory: 1 GB199.99
      Video card: Radeon 128MB49.99
      Sound Card: SoundBlaster Audigy 269.99
      LAN card: DLink 10/1004.99
  system cost = $536.94
```

This verifies correct construction of a PCComponent instance with one layer of subcomponents. It also verifies correct recursive construction of the data used by the toString() and getCost() methods.

Comment out line 33, and uncomment line 34. Run the program. The IDE's Output Window should show

```
Got data for PC System: Moby system24.99
      System enclosure: Gamer showoff case144.99
            Power Supply: Gamer 600W79.99
            System main board: Power User MB136.99
                  Processor: Athlon XP 240074.99
                  On-board Memory: 2 GB349.99
                  Video card: RenderMonster 384MB1859.99
                  Sound Card: SoundBlaster Audigy 269.99
                  LAN card: Trendware 10/100/100029.99
            Floppy Drive: Mitsumi All-Media34.99
            Hard Disk: Maxtor 160 GB104.99
            Optical Drive: Pioneer DVD RW85.99
      Keyboard: Avant Prime148.99
      Pointing Device: optical scrollmouse9.99
      Monitor: Sony 19-inch LCD599.99
      Printer: Epson Stylus Photo R800349.99
      External Speakers: Klipsch Promedia Ultra 5.1349.99
  system cost = $4456.83
```

This verifies correct construction of a top level PCComponent instance. It also verifies the greatest extent of recursive data construction for the toString() and getCost() methods.

Use Cases for the User Interface

(modified from a preliminary version by David Hostettler)

Use Case ID: UC1	**Use Case Name:** PC System Selection
Primary Actor(s):	The primary actor for this use case is the user of this deliverable.
Description:	A list of available PC systems will be displayed to the user. The user will then select a PC system to be viewed.
Preconditions:	The database must be available and populated with all required component and PC system information. The user must invoke the java executable to start this process.
Postconditions:	A list of available PC systems will be displayed.

Use Case ID: UC2	**Use Case Name:** PC System Detail Display
Primary Actor(s):	The primary actor for this use case is the user of this deliverable.
Description:	The user will select a system to view from a displayed list of PC systems. The selected PC system will then be displayed with all its components. A cost will be displayed next to each component. The total system cost will be displayed at the end of the component list.
Preconditions:	Successful execution of.UC1 The user must have selected a PC system from the displayed list.
Postconditions:	The PC system and all its components will be displayed with the cost of each component, and the sum of all costs.

Use Case ID: UC3	**Use Case Name:** Return to PC System Selection
Primary Actor(s):	The primary actor for this use case is the user of this deliverable.
Description:	To be able to choose another PC System for detail display, the user shall select an option that returns to the PC System Selection page.
Preconditions:	The PC System Detail Display page shall be shown.
Postconditions:	The PC System Selection page shall be shown.

Use Case ID: UC4	**Use Case Name:** Exit to Operating System
Primary Actor(s):	The primary actor for this use case is the user of this deliverable.
Description:	From a list of options, the user will choose to quit the program. The program shall inform the user that it is terminating, then exit to the operating system.
Preconditions:	The program shall display the PC System Selection page (see UC1).
Postconditions:	The database shall continue normal operation.

Integration / System Test

Start the database.

In the IDE, load **PcSystemViewer.java** into the editor.

Run the program. Execution should be as shown in **ProgramRun.doc**.

(I know – this is more properly called a regression test, since I'm comparing current operation to a previous successful example, rather than doing comparison to a set of specifications. At least it's a test.)

The Task 2 Problem Statement:

Create classes for the following types of PC components:

PCSystem
- Case
  - Power Supply
  - Motherboard
    - CPU
    - Memory
    - Video Card
    - Sound Card
    - Ethernet Card
  - Floppy Drive
  - Hard Drive
  - CD Drive
- Keyboard
- Mouse
- Monitor
- Printer
- Speaker

Each class should be composed of the components shown indented directly below it. Each class instance should have a data member with the name of the instance. For example, an instance of printer might have a name of "Epson Stylus 600" while another instance might have a name of "HP Deskjet 540." Also, each class should contain a member function, cost(), which returns the cost in dollars of the instance. In addition, each class should include a member function, toString (), which returns a nicely formatted string containing a recursive listing of the components making up a component.

Create database tables with your favorite relational database for the above component. Store some data in each table. Provide an interface to select data and display in a well formatted report. You may use Swing's JTable to store the data.