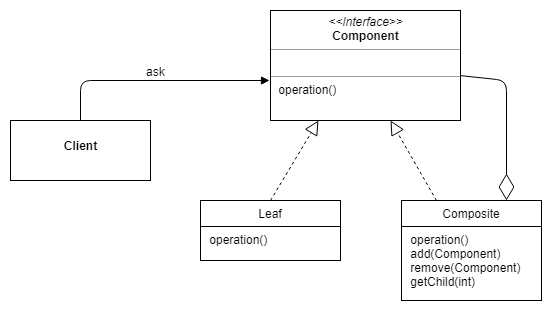
|  |  |
| --- | --- |
| **Project  Title** | **Composite Design Pattern** |
| **Author** | **Asim Ahmed (SP17-MSCS-0016)** |
| **Course** | **Design Patterns** |
| **Department** | **MS Computer Science** |

**Note: All the diagrams created from https://www.draw.io/**

**Intent**

Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

**Abstract Diagram**

****

**Participants**

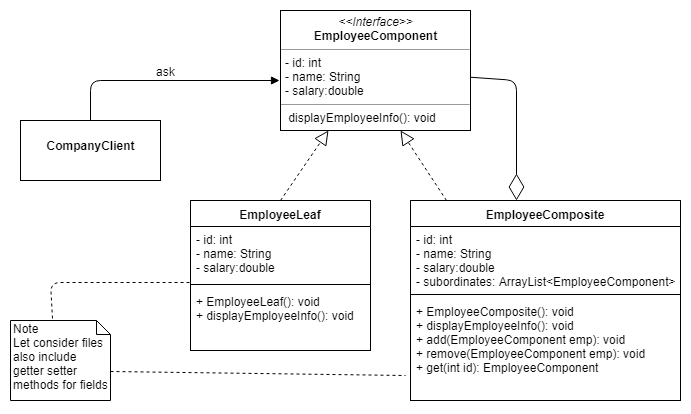
* **Component –**Component declares the interface for objects in the composition and for accessing and managing its child components. It also implements default behavior for the interface common to all classes as appropriate.
* **Leaf –**Leaf defines behavior for primitive objects in the composition. It represents leaf objects in the composition.
* **Composite –**Composite stores child components and implements child related operations in the component interface.
* **Client –** Client manipulates the objects in the composition through the component interface.

**-------------------------------------------------------- XXXXXXXXXXXXXXXXXXXXXXX------------------------------------------------------**

**Real Life Examples**

**Example-1**

This example is about an organization, which has employees with different roles. It has general managers and under general managers, there can be managers and under managers there can be developers. Now you can set a tree structure and ask each node to perform common operation like displaying employee detail. In this example developer act as leaf node of design pattern, general managers and managers’ act as composite node of design pattern, so there will be one interface EmployeeComponent which is be implemented by EmployeeComposite and EmployeeLeaf. Company acts as client for which CompanyClient is used.

**Class Diagram**

**EmployeeComponent.java**

|  |
| --- |
| public interface EmployeeComponent{      private int id;  private String name;      private double salary;      public void displayEmployeeInfo();  } |

**EmployeeLeaf.java**

|  |
| --- |
| public class EmployeeLeaf implements EmployeeComponent {      private int id;  private String name;      private double salary;    // Let consider class also include getter setter for fields      public EmployeeLeaf (int id, String name, double salary){          this.id = id;          this.name = name;          this.salary = salary;      }        @Override      public void displayEmployeeInfo() {          System.out.println("ID:" + id);          System.out.println("Name: " + name);          System.out.println("Salary:" + salary);      }  } |

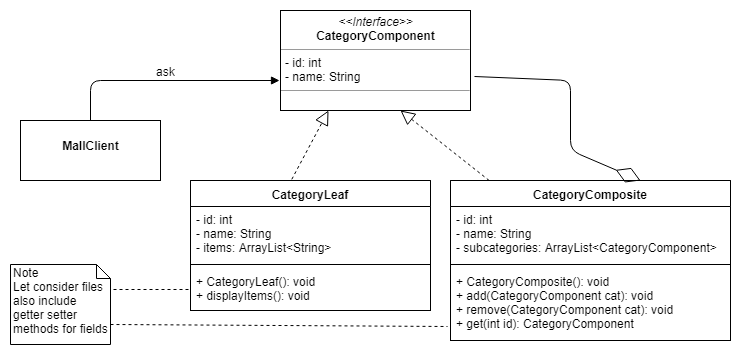
**EmployeeComposite.java**

|  |
| --- |
| public class EmployeeComposite implements EmployeeComponent {      private List< EmployeeComponent > subordinates = new ArrayList< EmployeeComponent >();      private int id;  private String name;      private double salary;    // Let consider class also include getter setter for fields      public EmployeeComposite(int id, String name, double salary){          this.id = id;          this.name = name;          this.salary = salary;      }        @Override      public void displayEmployeeInfo() {          System.out.println("ID:" + id);          System.out.println("Name: " + name);          System.out.println("Salary:" + salary);      }        public void add(EmployeeComponent emp){          employeeList.add(emp);      }        public void removeRemove(EmployeeComponent emp){          employeeList.remove(emp);      }    public EmployeeComponent get(int id){          foreach(EmployeeComponent emp: items){  If(emp.getId() == id){  return emp;  }  return null;  }  } |
|  |

**CompanyClient.java**

|  |
| --- |
| public class CompanyClient{      public static void main (String[] args){          EmployeeLeaf dev1 = new Developer(100, "Asad", 80000          EmployeeLeaf dev2 = new Developer(101, "Zarwalli", 40000);          EmployeeComposite lead = new EmployeeComposite(200, "Asim", 100000);          EmployeeComposite sdm = new Manager(50, "Taha ", 200000);            lead.add(dev1);          lead.add(dev2);        sdm.add(lead);  sdm.displayEmployeeInfo();  lead.displayEmployeeInfo();        }  } |
| **-------------------------------------------------------- XXXXXXXXXXXXXXXXXXXXXXX-------------------------------------------**  **Example-2**  This example is from retail system and is about Category hierarchies, in which inventory item belongs to a category, and category has subcategory hierarchies depends on the organization (clubs, restaurants, malls) setup. In this example Category acts as Component, Composite and Leaf entity of design pattern and Mall is client. Items list belongs to leaf category. CategoryComponent is an interface which is implemented by CategoryLeaf and CategoryComposite. MallClient class is used for client. |

**Class Diagram**

****

**CategoryComponent.java**

|  |
| --- |
| public interface CategoryComponent{      private int id;  private String name;  } |

**CategoryLeaf.java**

|  |
| --- |
| public class CategoryLeaf implements CategoryComponent {      private int id;  private String name;  ArrayList<String> items = new ArrayList<String();    // Let consider class also include getter setter for fields      public CategoryLeaf (int id, String name, ArrayList<String> items){          this.id = id;          this.name = name;          this.items = items;      }        public void displayItems() {  foreach(String s : items){  System.out.println("Item Name: " +s);  }  }  } |
|  |

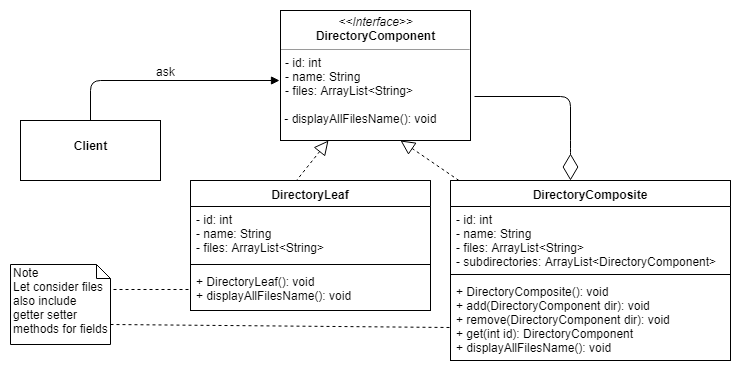
**CategoryComposite.java**

|  |
| --- |
| public class CategoryComposite implements CategoryComponent {      private List< CategoryComponent > subcategories = new ArrayList< CategoryComponent >();      private int id;  private String name;    // Let consider class also include getter setter for fields      public CategoryComposite(int id, String name){          this.id = id;          this.name = name;      }        public void add(CategoryComponent emp){          subcategories.add(emp);      }        public void removeRemove(CategoryComponent emp){          subcategories.remove(emp);      }    public CategoryComponent get(int id){          foreach(CategoryComponent cat: subcategories){  If(cat.getId() == id){  return cat;  }  return null;  }      }  } |
|  |

**MallClient.java**

|  |
| --- |
| public class MallClient{      public static void main (String[] args){  ArrayList<String> items = new ArrayList<String();  Items.add(“Golf Ball”);  Items.add(“Tennis Ball”);  Items.add(“Cricket Ball”);          CategoryLeaf cat1 = new CategoryLeaf (101, "Balls", items);          CategoryComposite cat2 = new CategoryComposite(200, "Accessories");          CategoryComposite cat3= new CategoryComposite (50, "Mild Balls");            cat2.add(cat1);          cat3.add(cat2);  cat1.displayItems();      }  } |
| **-------------------------------------------------------- XXXXXXXXXXXXXXXXXXXXXXX-------------------------------------------**  **Example-3**  This example is about directory hierarchies, such as operating systems have. It is simple example in which a system has directories (folders) and directory contains other directories and each directory can have files too. In this example directory acts as component, composite and leaf entity of design pattern. Composite and leaf classes both can have files. DirectoryComponent is an interface which is implemented by DirectoryComposite and DirectoryLeaf. |

**Class Diagram**

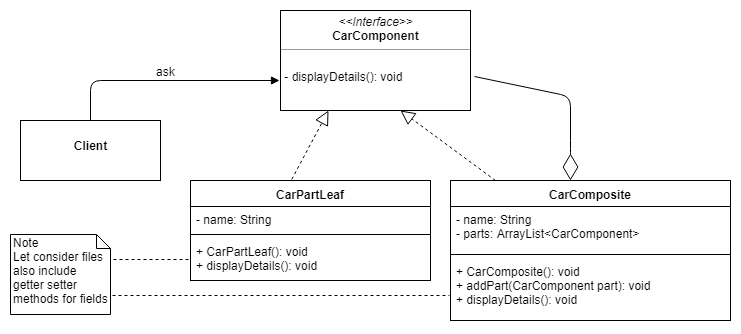


**-------------------------------------------------------- XXXXXXXXXXXXXXXXXXXXXXX------------------------------------------------------**

**Example-4**

This example is about building a car. Consider we want to build car with specific parts. Each part has its own existence and it can be added to a car. We want to perform some operation on car and want it to be done on all its individual objects. In this example CarComponent is an interface which has abstract method about to display detail. CarPart class acts as leaf node and implements CarComponent. Finally CarComposite class act as composite node and it holds a list of individual elements and a way to add them into existing list and it also implement CarComponent.

**Class Diagram**

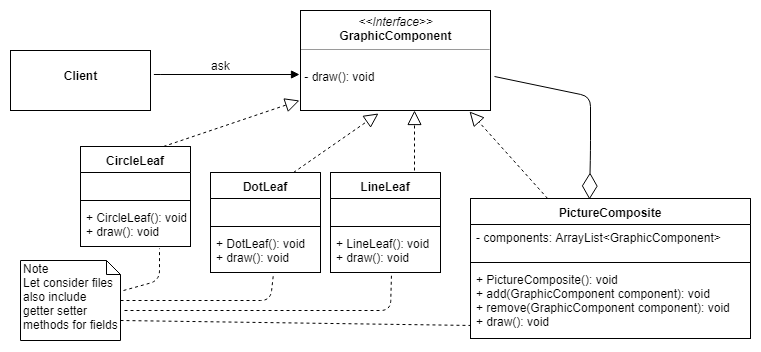


**-------------------------------------------------------- XXXXXXXXXXXXXXXXXXXXXXX------------------------------------------------------**

**Example-5**

This example is about graphics applications like drawing editors that let users build complex diagrams out of simple components. The user can group components to form larger components, which in turn can be grouped to form still larger components. A simple implementation could define classes for graphical primitives such as Text and Lines plus other classes that act as containers for these primitives. So in this example GraphicComponent act as component interface of design pattern and it contains draw method. PictureComposite act as the composite class of design pattern and it contains list of other graphic components which are leaf classes of design pattern can be circle, line, dot, and rectangle and so on.

**Class Diagram**



**-------------------------------------------------------- XXXXXXXXXXXXXXXXXXXXXXX------------------------------------------------------**