Assignment 1

public abstract class Course

{

public void start()

{

// start process

}

public abstract double getFee();

public abstract void evaluate();

}

define child classes of above abstract class

DACDBDACourse extends Course

MSCitCourse extends Course

show late binding with the help of "perform()" method.

public abstract class Course{

public void start() {

System.***out***.println("Course class start() method");

}

public abstract double getFee();

public abstract void evaluate();

}

class DACDBDACourse extends Course{

public void start() {

System.***out***.println("DACDBDA Course started");

}

public double getFee() {

return 135700;

}

public void evaluate() {

System.***out***.println("Evaluated DACDBDA course");

}

}

class MSCitCourse extends Course{

public void start() {

System.***out***.println("MSCIT Course started");

}

public double getFee() {

return 4000;

}

public void evaluate() {

System.***out***.println("Evaluated MSCIT Couse ");

}

}

public class Demo{

static void perform(Course c) {

System.***out***.println("Fees of this course is: "+ (c.getFee()));

c.evaluate();

c.start();

}

public static void main(String [] args) {

*perform*(new DACDBDACourse());

*perform*(new MSCitCourse());

}

}

Assignment 2

abstract class Beverages {

// Defined

public void addMilk() {

}

// abstract

abstract void addIngredients();

}

class Coffee extends Beverages

{

void addIngredients()

{

// ingredients for Coffee

}

// addMilk inherited

}

class Tea extends Beverages

{

void addIngredients()

{

// ingredients for Tea

}

// addMilk inherited

}

class FruitJuice extends Beverages

{

void addIngredients()

{

// ingredients for FruitJuice

}

// addMilk inherited

}

create Array of Beverages class and store all the child classes objects.

now traverse the array and invoke all the functions.

public abstract class Beverages {

public void addMilk() {

System.***out***.println("Milk added");

}

abstract void addIngredients();

}

class Coffee extends Beverages{

public void addMilk() {

System.***out***.println("Milk added");

}

void addIngredients() {

System.***out***.println("Added coffee");

}

}

class Tea extends Beverages{

public void addMilk() {

System.***out***.println("Milk added");

}

void addIngredients() {

System.***out***.println("Added Tea powder");

}

}

class FruitJuice extends Beverages{

public void addMilk() {

System.***out***.println("Milk added");

}

void addIngredients() {

System.***out***.println("Added Fruits");

}

}

public class Demo {

public static void main(String[] args) {

// **TODO** Auto-generated method stub

Beverages [] b = new Beverages[3];

b[0] = new Tea();

b[1] = new Coffee();

b[2] = new FruitJuice();

for(Beverages ref : b) {

ref.addIngredients();

ref.addMilk();

}

for(int i=0; i<b.length;i++) {

b[i].addIngredients();

b[i].addMilk();

}

}

}

Assignment 3

create abstract class "Shape" with

a contract "draw()" and concrete behaviour "show()"

create 3 child classes

Triangle, Rectangle and Circle

define a class "ShapeDemo" with main

inside main create an array of "Shape" of 3 elements. Store all the child class objects inside the array, traverse it and invoke "draw()" and "show()" methods polymorphically.

public abstract class Shape {

public void show() {

System.***out***.println("Shape shown");

}

abstract void draw();

}

class Triangle extends Shape{

public void draw() {

System.***out***.println("Triangle drawn");

}

public void show() {

System.***out***.println("Triangle Shown");

}

}

class Rectangle extends Shape{

public void draw() {

System.***out***.println("Rectangle drawn");

}

public void show() {

System.***out***.println("Rectangle Shown");

}

}

class Circle extends Shape{

public void draw() {

System.***out***.println("Circle drawn");

}

public void show() {

System.***out***.println("Circle Shown");

}

}

public class ShapeDemo {

public static void main(String[] args) {

// **TODO** Auto-generated method stub

Shape [] s = new Shape[3];

s[0] = new Triangle();

s[1] = new Rectangle();

s[2] = new Circle();

for(Shape ref : s) {

ref.draw();

ref.show();

}

}

}

Assignment 4

public class UIComponent

{

}

public class Button extends UIComponent

{

}

public class TextField extends UIComponent

{

}

public class CheckBox extends UIComponent

{

}

public abstract class UIComponentCreator

{

public void show(String type)

{

UIComponent comp=createUIComponent(type);

add(comp);

}

public void add(UIComponent component)

{

System.out.prinlnt("Added the component\t"+component);

}

public abstract UIComponent createUIComponent(String type);

}

create 3 child classes of "UIComponentCreator"

WindowsUIComponentCreator, LinuxUIComponentCreator and MacUIComponentCreator

in all these classes "createUIComponent()" method must be defined in such a way that whatever type is passed as per this it must return a specific UIComponent.

e.g. if "button" is passed , it should return "Button" class object or if "textfield" is passed , it should return "TextField" class object.

create a class UIComponentCreatorDemo with main function

inside main function

create objects of WindowsUIComponentCreator, LinuxUIComponentCreator and MacUIComponentCreator classes and invoke "createUIComponent()" ,"add()"

and "show()" methods.

public abstract class UIComponentCreator {

public abstract UIComponent createUIComponent(String type);

public void show(String type) {

UIComponent comp = createUIComponent(type);

add(comp);

}

void add(UIComponent component) {

System.***out***.println("The UI Component is : "+ component);

}

}

public class WindowsUIComponentCreator extends UIComponentCreator {

public UIComponent createUIComponent(String type) {

// **TODO** Auto-generated method stub

if(type.equalsIgnoreCase("Button")) {

return new Button();

}

if(type.equalsIgnoreCase("TextField")) {

return new TextField();

}

if(type.equalsIgnoreCase("CheckBox")) {

return new CheckBox();

}

return null;

}

}

public class MacUIComponentCreator extends UIComponentCreator {

public UIComponent createUIComponent(String type) {

// **TODO** Auto-generated method stub

if(type.equalsIgnoreCase("Button")) {

return new Button();

}

if(type.equalsIgnoreCase("TextField")) {

return new TextField();

}

if(type.equalsIgnoreCase("CheckBox")) {

return new CheckBox();

}

return null;

}

}

public class LinuxUIComponentCreator extends UIComponentCreator {

public UIComponent createUIComponent(String type) {

// **TODO** Auto-generated method stub

if(type.equalsIgnoreCase("Button")) {

return new Button();

}

if(type.equalsIgnoreCase("TextField")) {

return new TextField();

}

if(type.equalsIgnoreCase("CheckBox")) {

return new CheckBox();

}

return null;

}

}

public class UIComponentCreatorDemo {

public static void main(String[] args) {

// Windows creator

UIComponentCreator win = new WindowsUIComponentCreator();

UIComponent wbtn = win.createUIComponent("button");

win.add(wbtn);

win.show("textfield");

// Linux creator

UIComponentCreator lin = new LinuxUIComponentCreator();

UIComponent lcb = lin.createUIComponent("checkbox");

lin.add(lcb);

lin.show("button");

// Mac creator

UIComponentCreator mac = new MacUIComponentCreator();

UIComponent mtf = mac.createUIComponent("textfield");

mac.add(mtf);

mac.show("checkbox");

}

}

Assignment 5 command line

accept 3 numbers through command line arguments and display their sum.

public class Demo {

public static void main(String[] args) {

// **TODO** Auto-generated method stub

int[] arr = new int[3];

arr[0] = Integer.*parseInt*(args[0]);

arr[1] = Integer.*parseInt*(args[1]);

arr[2] = Integer.*parseInt*(args[2]);

System.***out***.println("sum is : "+(arr[0]+arr[1]+arr[2]));

}

}