

**Course Code**: CSE110

**Section No**: 01

**Submitted To:**

Tanni Mittra

Senior lecturer

Department of CSE

**Submitted By:**

**Student’s ID :** 2020-2-60-112

**Student’s Name** : Md Tahsin

**Problem 1(a):**

package Mid2Assignment;

abstract class Sequence {

double number;

Sequence(){

this.number=0;

}

Sequence(double number){

this.number=number;

}

public void setnumber(double number){

this.number=number;

}

public double getnumber(){

return number;

}

abstract int Term(int n);

abstract void toString(int n);

}

**Justification:**

An instance of Sequence class can’t be created, because it’s an abstract class which is not fully implemented. Abstract class is a template where both abstract method and non-abstract method can be used, abstract method can be implemented in other non-abstract class which is the subclass of the abstract class, therefore, abstract class is not fully implemented.

**Problem 1(b):**

package Mid2Assignment;

import java.util.\*;

public class MySequence extends Sequence {

MySequence(){

}

MySequence(int n){

super(n);

}

@Override

public int Term(int n) {

if (n == 0 || n == 1) {

return n;

}

int z = Term(n - 1) + Term(n - 2);

return z;

}

@Override

public void toString(int n){

for(int i=2;i<=n+1;++i){

if(i<n-1){

System.out.print(Term(i)+",");

}

else{

System.out.print(Term(i)+",");

}

}

}

public static void main(String args[]){

Scanner in=new Scanner (System.in);

System.out.println("Input the value:");

int a= in.nextInt();

MySequence obj=new MySequence();

obj.toString(a);

obj.Term(a);

}

}

**Problem 1(c):**

package Mid2Assignment;

import java.util.Scanner;

class Series extends Sequence {

@Override

int Term(int n) {

int x=0;

int y=1;

int z=1;

for(int i=3; i<=n; i++){

z=x+y;

x=y;

y=z;

}

}

@Override

void toString(int n) {

for (int i=0;i<n;i++){

System.out.println(((n-1)+number)\*n);

}

return super.toString();

}

void Sum(int n){

int x=0,y=1,c,s=0,i;

for ( i=2;i<n;i++){

c=x+y;

s+=c;

x=y;

y=c;

}

System.out.print("Sum: ");

System.out.println(s+1);

}

String sumString(int n){

for (int i=0;i<n;i++){

return "t1: ";

}

return "t1";

}

public static void main(String args[]){

Scanner in=new Scanner (System.in);

System.out.println("Input the value:");

int a= in.nextInt();

Series obj=new Series();

obj.Sum(a);

}

}

**Problem 2(a):**

package Mid2Assignment\_2;

public interface Command {

String buffer = "de";

StringBuffer doc = new StringBuffer();

void Execute();

}

class Save implements Command {

String n;

public Save(String n) {

this.n = n;

}

@Override

public void Execute() {

doc.append(n);

}

}

**Problem 2(b):**

class Copy implements Command{

int a, b;

String remove;

public Copy(int a, int b) {

this .a = a;

this .b = b;

}

public void Cut() {

remove = Command.doc.substring(a, b);

Command.doc.delete(a, b);

}

public void undo() {

Command.doc.insert(a, remove);

}

@Override

public void Execute() {

}

}

**Problem 2(c):**

package Mid2Assignment\_2;

class Paste implements Command {

int a;

int pasted\_size;

Paste(int a) {

this.a = a;

}

@Override

public void Execute() {

Command.doc.insert (a, Command.buffer );

pasted\_size = Command.buffer.length ();

}

void undo() {

Command.doc.delete(a,a+pasted\_size);

}

}

**Problem 2 Test:**

package Mid2Assignment\_2;

public class test {

public static void main(String[] args) {

Command save = new Save("abcdefgh");

save.Execute();

System.out.println(Command.doc);

Copy copy = new Copy(3,5);

copy.Cut();

System.out.println(Command.doc);

Paste paste = new Paste(0);

paste.Execute();

System.out.println(Command.doc);

paste.undo();

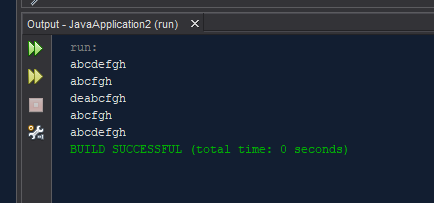
System.out.println(Command.doc);

copy.undo();

System.out.println(Command.doc);

}

}



**Problem 3(a):**

Yes, fruit is instanceof GoldenDelicious.

**Justification:**

package Mid2Assignment\_3;

public class Fruit {

}

class Apple extends Fruit {

}

class Orange extends Fruit {

}

class GoldenDelicious extends Apple {

}

class McIntosh extends Apple {

}

public class test {

public static void main(String[] args ){

Fruit fruit = new GoldenDelicious();

Orange orange = new Orange();

if(fruit instanceof GoldenDelicious){

System.out.println("Yes");

}

else{

System.out.println("No");

}

}

}

**Problem 3(b):**

Yes, orrange can invoke makeorrangejuice mathod.But fruit can't

invoke that method.

If we define the method makeOrangeJuice in the Orange class then the class can be invoked the method. Because makeOrangeJuice is an instance of Orange class.On the other hand, fruit cannot invoke this method. Because object fruit is not an instance of Orange class.

**Problem 3(c):** No, that statement is not legal.

**Problem 4(a):**

An interface is also a type.

In general interface is the collections of abstract methods. Interface is also like a class which can contain methods and can be written in “.java” extension. Therefore, we can say “An interface is also a type”.

**Problem 4(b):**

package Mid2Assignment\_4b;

import java.util.\*;

public class Person implements Comparable<Person> {

private String firstName;

private String lastName;

private int age;

Person() {

}

Person(String firstName, String lastName, int age) {

this.firstName = firstName;

this.lastName = lastName;

this.age = age;

}

public String getfirstname() {

return firstName;

}

public String getlastname() {

return lastName;

}

public int getage() {

return age;

}

@Override

public int compareTo(Person y) {

Person x = (Person) y;

if (getlastname().compareTo(y.getlastname()) == 0) {

return getfirstname().compareTo(y.getfirstname());

}

else if (getfirstname().compareTo(y.getfirstname()) == 0) {

return getlastname().compareTo(y.getlastname());

} else {

return getage() > y.getage() ? 1 : getage() < y.getage() ? -1 : 0;

}

}

//Testing

public static void main(String[] args) {

Scanner inp = new Scanner(System.in);

Person obj1 = new Person();

obj1 = new Person("Logan", "paul", 33);

System.out.print(" Comparing with Nmae: Logan paul \n Age: 33\n");

System.out.println(" Enter new entry :");

System.out.println(" Enter first name:");

String firstname = inp.next();

System.out.println(" Enter last name:");

String lastname = inp.next();

System.out.println(" Enter age:");

int a = inp.nextInt();

Person obj2 = new Person(firstname, lastname, a);

obj2.toString();

int result = obj2.compareTo(obj1);

System.out.println("Successfully compared");

System.out.println(obj1.compareTo(obj2));

if (obj1.compareTo(obj2) < 0) {

System.out.println("not matched");

} else if (obj1.compareTo(obj2) > 0) {

System.out.println("not matched");

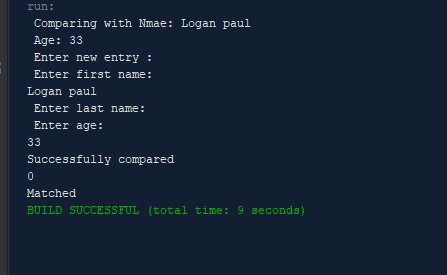
} else {

System.out.println("Matched");

}

}

}

****

**Problem 4(c):**

Interface inconsistency refers the implementation of two same methods with exact functions, but does the different task. So, the elements and methods are look like same, but the tasks are different.

**Problem 4(d):**

The difference between an abstract class and an interface :

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| * Can have abstract and non-abstract methods. | * Can only have abstract methods. |
| * It’s doesn’t support multiple inheritance. | * It supports multiple inheritance. |
| * Can have static, non-static, final and non-final variable. | * Can only have static and final variables. |
| * Partial abstraction possible. | * Fully abstraction possible. |
| * An abstract class can extend another Java class and implement multiple Java interfaces. | * An interface can extend another Java interface only. |
| * Example:  abstract class Animal{  abstract void Eat(); } | * Example: interface Animal{ void Eat(); } |

**Problem 4(e):**

* An interface can be used to define an agreement behavior and it can also act as an agreement to interact between classes. Besides, an abstract class is basically used to define the default behavior for a subclass, all child classes have the same functionality. In such case, I would choose interface rather than abstract class.
* An abstract class allows to create functionality that can apply or override subclasses. Besides an interface allows to define functionality, not apply it. And where a class can only expand an abstract class, it can take advantage of multiple interfaces. In that case, I would choose abstract class rather than interface.