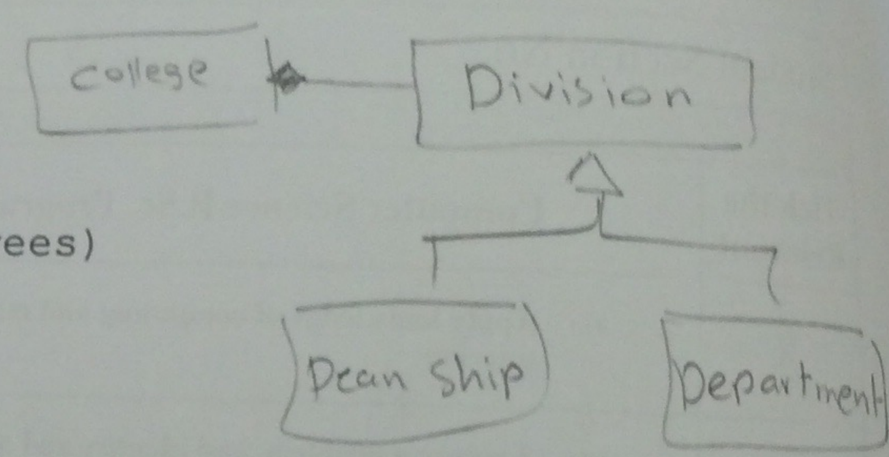
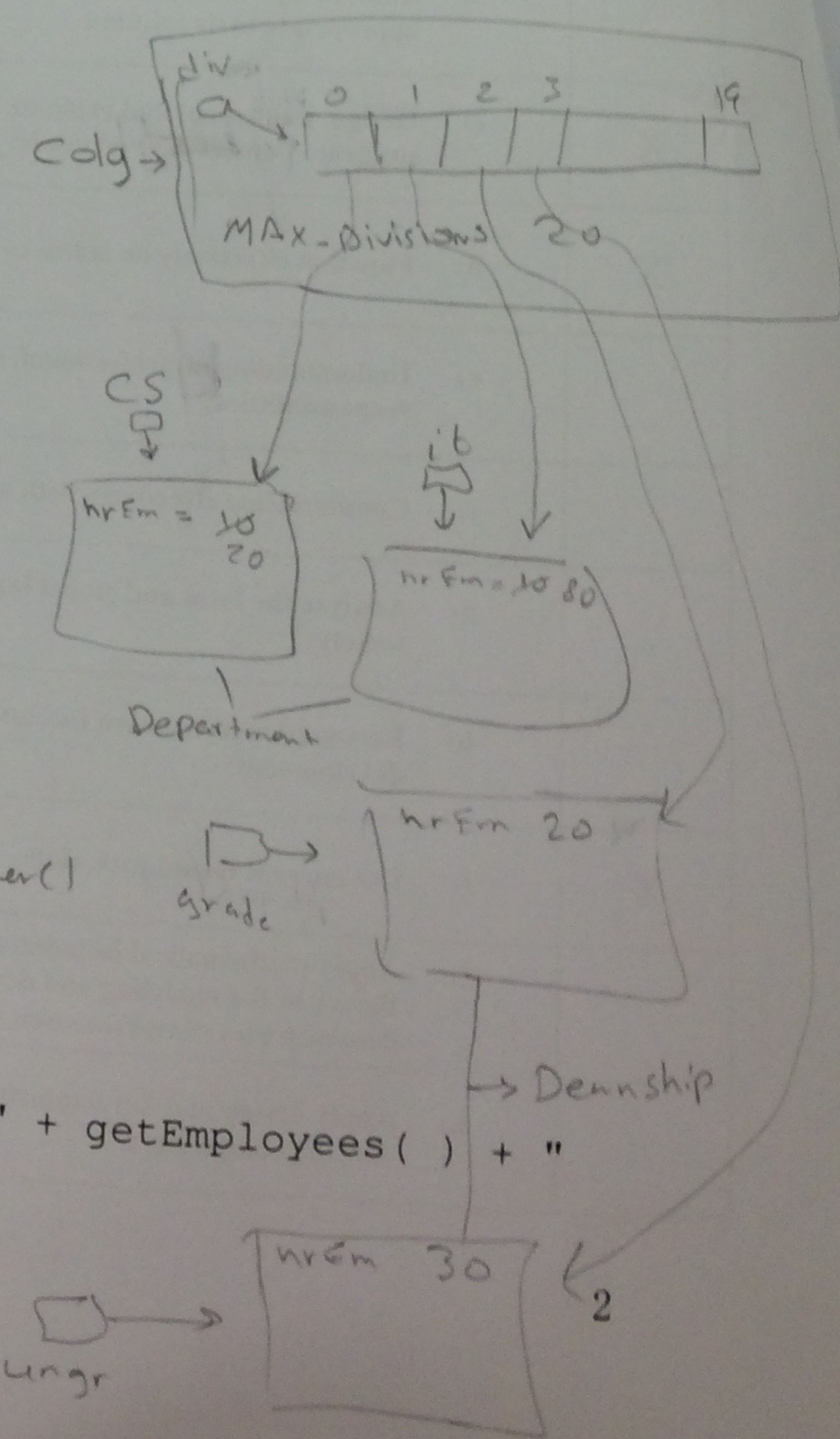
Question 1: **What is the output of the following program [5 Marks]?**

```
public class Divisions
{
   private int nrEmployees;

   public Divisions()
   { this( 10 );}

   public Divisions (int nrEmployees)
   { setEmployees(nrEmployees); }

   public String toString ()
   {return this.getClass().getName()+ " with " + getEmployees()+ "
employees";}

   public int getEmployees ()
   { return nrEmployees; }

   public void setEmployees (int employees)
   { nrEmployees = employees; }
}
```



```
public class DeanShip extends Divisions
{
   public DeanShip ()
   { this(10); }

   public DeanShip(int employees)
   { super(employees); }
}
```



```
public class Department extends Divisions
{
   public Department ()
   { super(10); }

   public Department (int nrEmployees)
   { setEmployees(nrEmployees); }

   public String toString()
   {
      return this.getClass().getName()+" has " + getEmployees( ) + "
employees";}
```

output:

```
}

public class College
{
   private final static int MAX_DIVISIONS =20;
   private int nrDivisions;
   private Divisions [] divs;


public College ()
   {
      nrDivisions = 0;
      divs = new Divisions[MAX_DIVISIONS];
   }

   public int nrAddedDiv (  )
   { return nrDivisions; }
                                  cs, it, gra, ung
   public void add (Divisions d )
   { divs [nrDivisions ++ ] = d; }

   public int totalEmployees ( )
   {
      int nrEmployees = 0;
      for (int d = 0; d < nrDivisions; d++ )
         nrEmployees += divs [ d ].getEmployees( );
      return nrEmployees;
   }

   public String toString( )
   {
      String s = "";              4
      for (int d = 0; d < nrDivisions; d++ )
         s += divs[d].toString( ) + "\n";
      s += "Total number of employees for all divisions " +
totalEmployees();
      return s;
   }}
```

```
public class Exam1Code
{
   public static void main ( String[ ] args )
   {
      College colg= new College();
      Department cs = new Department( 20 );
      Department it = new Department( 80 );
      DeanShip graduate = new DeanShip ( 20 );
      DeanShip undergraduate = new DeanShip (30);
      colg.add ( cs );
      colg.add ( it );
      colg.add ( graduate );
      colg.add ( undergraduate );
      System.out.println(colg.toString( ) );
 }}
```

Output:

2/5

```
Department has 20 employees
Department has 80 employees
Deanship with 20 employees
Deanship with 30 employees
```

Question 2: Programming Problem [20 Marks]

Given the following UML and the corresponding classes descriptions, do the following:

1- Write the Java code for the ***MainCourse***, and ***Restaurant*** classes

Note: suppose that all Setters and Getters are provided.

2- Write a Test class that does the following:

a. Create 1 **Dessert** item and 1 **MainCourse** item of your choice.

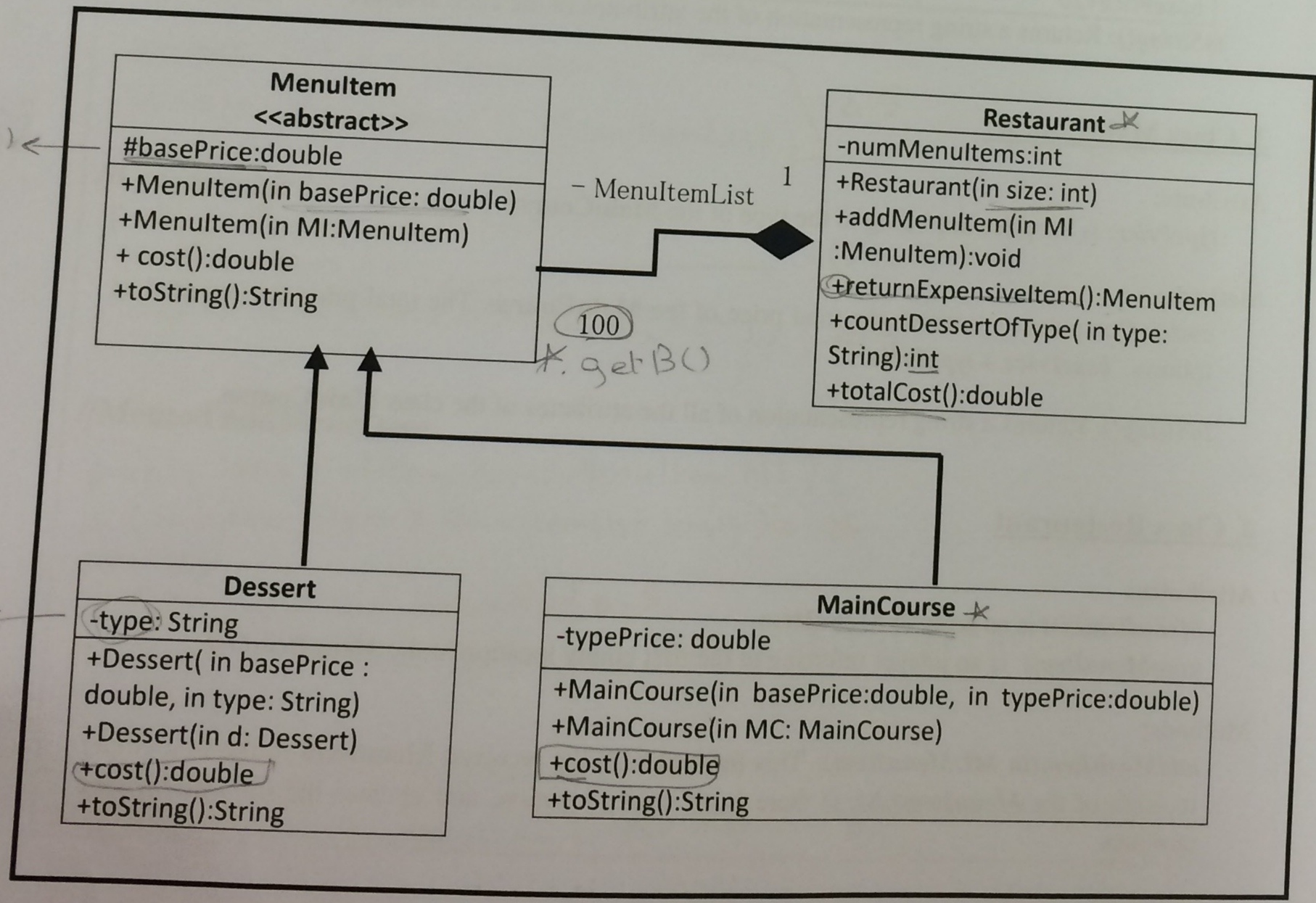
b. Calculate and display the cost of each item created in step (a).

c. Create an object of type **Restaurant** containing the items you created in step (a).

d. Compute and display the total cost of all the items in the **Restaurant**.

e. Display the kind (**MainCourse** or **Dessert**) of the most expensive item in the **Restaurant**, and also display its information.

f. Count and display the number of ***Cake*** Desserts in the **Restaurant**.



1. Class MenuItem

Attribute:

basePrice is the base price of the **MenuItem**

Methods:

cost(): **An abstract method** that computes and returns the total price of the **MenuItem**

toString(): Returns a string representation of all the attributes of the class **MenuItem**

2. Class Dessert

Attribute:

type: Specifies the type of the **Dessert**

Methods:

cost(): Computes and returns the total price of the **Dessert**. The total price of the **Dessert** depends on its ***type*** according to the following table

Cake	Tart	Ice Cream
basePrice+20	basePrice+30	basePrice+15

toString(): Returns a string representation of the attributes of the class **Dessert**

3. Class MainCourse:

Attribute:

typePrice: is the price according to the type of the **MainCourse**

Methods:

cost(): Computes and returns the total price of the **MainCourse**. The total price is computed as follows: ***basePrice + typePrice***

toString(): Returns a string representation of all the attributes of the class **MainCourse.**

4. Class Restaurant

Attributes:

MenuItemList is an array of **MenuItem**.

numMenuItems is an integer referring to the first empty location in the ***MenuItemList***

Methods:

addMenuItem(in MI:MenuItem): This method adds the received **MenuItem** in the first empty location of the ***MenuItemList***, if there is an empty location, and updates the ***numMenuItems*** attribute.

returnExpensiveItem(): Returns a copy of the item that has the highest ***basePrice*** in the ***MenuItemList***.

countDessertOfType (in type:String): Counts the number of **Desserts** having the given ***type*** in the ***MenuItemList*** and returns the result as an integer.

totalCost(): Computes the total cost of the items in the ***MenuItemList*** and returns the result as a double.

Note: use casting only when necessary

length

2 > 3

2 > 2 ×

0 1 2

1 2 3

11.75 / 12

Class Restaurant:

//Class Header

```
Class Restaurant {
```

//attributes

```
private int numMenuItem;
private MenuItem [] MenuItemList;
```

0.5

//Constructor

```
public Restaurant (int size){
numMenuItem = 0;
MenuItemList = new MenuItem[size]; }
```

if (size < 101)

0.5

//Method addMenuItem

```
public void addMenuItem( MenuItem MI ){
if (numMenuItem > MenuItemList.length )
return;
if (MI instanceof Dessert){
MenuItemList[numMenuItem] = new Dessert((Dessert)(MI));
numMenuItem++;
return; }
if (MI instanceof MainCourse){
MenuItemList[numMenuItem] = new MainCourse((MainCourse)(MI));
numMenuItem++;
return; }}
```

0.25 0.5 1 0.25 1 0.25

2 > 2

1 > 1

check *num

```
if(numMenuItem == 0){
return null;}
```

Name			
ID		Serial #	
		Section	

//Method returnExpensiveItem

```
public MenuItem returnExpensiveItem(){
int j=0;  0.5
for(int i=1; i<numMenuItem; i++){ 0.25
if(MenuItemList[i].getB() > MenuItemList[j].getBasePrice()) 0.5
j = i; } 0.25
if(MenuItemList[j] instanceof Dessert){ 0.5
MenuItem M1 = new Dessert((Dessert)(MenuItemList[j]))
return M1; }
```
0.75

//Method countDessertOfType

```
Public int countDessertOfType(String Type){
int c=0; 0.25
for(int i=0; i<numMenuItem; i++){ 0.25
if(MenuItemList[i] instanceof Dessert){ 0.5
((Dessert)(MenuItemList[i])).setT().equals(Type)) 1
c++;} } 0.25
return c; } 0.25
```

//Method totalCost

```
public double totalCost(){
double sum=0; 0.25
for(int i=0; i<numMenuItem; i++){ 0.25
sum += MenuItemList[i].cost(); } 0.5
return sum; } 0.25
```

```
if(numMenuItem==0)
return 0;
```

```
if(MenuItemList[j] instanceof MainCourse){ 0.25
MenuItem M2 = new MainCourse((MainCourse)(MenuItemList[j]));
return M2; }
return null; } //Method
```
0.75

Name		Serial #	
ID		Section	

Class MainCourse:

(3)

//Class Header

```
class MainCourse extends MenuItem {
```

//attributes

```
private double typePrice;
```

//Constructors

```
public MainCourse(double basePrice, double typePrice){
super(basePrice);
this.typePrice = typePrice; }

public MainCourse(MainCourse MC){
super(MC.getB());
typePrice = MC.typePrice; }
```

//Method cost

```
public double cost(){ return basePrice + typePrice ; }
```

//Method toString

```
public string toString(){ return super.toString() + " type:" + typePrice ; }
```

Name		Serial #	
ID		Section	

Test Class:

4.5

```
public class TestRestaurnt
{
    public static void main (String []args)
    {
      // (a)
      Dessert D = new Dessert(20, "Ch");
      Main Course C = new Main Course (20,40);

      // (b)
      System.out.print("cost of Dessert" + D.cost());
      system.out.print(" cost of Main" + C.cost());

      // (c)
      Resturant R = new Resturant(100);
      R.addMenuItem(D);
      R.addMenuItem(C);

      // (d)
      System.out.print("the total" + R.totalCost());

      // (e)
      if(R.returnExpensiveItem instanceof Dessert){
      S.O.P((Dessert)(R.returnExpensiveItem)).tostring());
      S.O.P("it's Dessert"); }
      else if (R.returnExpensiveItem instanceof Main Course){
      S.O.P(R.returnExpensiveItem.tostring)
      S.O.P("it's main Course"); }

      // (f)
      S.O.P("cake:" + R.CountDessertOfType("cake"));
}
}
```

Type must be for table -0.25

no need for cast -0.25