4(c). Consider the following football clubs, their points and position in the English Premier League: (5)

Club	Point	Position
Manchester United	90	1
Chelsea	80	2
Arsenal	75	4
Manchester City	30	19
Liverpool	25	20

Write Java code for the following:

i. Declare a single Enumeration named EPL for the above clubs
ii. Write Java code to print all the clubs with their point and position.

```
i) enum EPL {
ManchesterUnited("Manchester United", 90, 1),
Chelsea("Chelsea", 80, 2),
Arsenal("Arsenal", 75, 4),
ManchesterCity("Manchester city", 30, 19),
Liverpool("Liverpool", 25, 20);

    privat int name;
    private int point;
    private int position;

public int getPoint() {return point;}
public int getPositionc() {return position;}

public void setPoint(int point)
    { this.point = point;}
public void setPosition(int position)
```

```
{ this. position = position ; }

EPL (String name, int point, int position)
{
    this. name = name;
    this . point = point;
} this. position = position;

public void SetName (String name)
{ this. name = name ; }

public String getName() { return name; }

@Override
public String toString() {
return "Club : " + name +
        "Point : " + point +
        "Position : " + position;

}
```

ii)

```
public class EnumDemo {
public static void main (String[]args){
  EPL epL[] = EPL.values();
  for (EPL e : epL)
System.out.println (e);
  }
}
```

(d) Consider the following football clubs, their point and position in the English Premier League: **(5)**

Club	Point	Position
ManUnited	90	1
Chelsea	80	2
Arsenal	75	4
Liverpool	30	19
ManCity	25	20

same as before

Write Java code for the following:

(i) Declare a single Enumeration named **EPL** for the above clubs

(ii) Find out the difference between league position of **ManUnited** and **Liverpool** (you can't simply write 18).

ii)

```
EPL m = EPL.ManUnited;
EPL l = EPL.Liverpool;

int diff = Math.abs
        (m.getPosition() - l.getPosition());
```

(b) Write three differences between Hashmap and Hashtable. Consider the following monthly salary of different cricketers: **(3×2+10=16)**

Contd P/3

HashMap | HashTable

Non-Synchronized	Synchronized
Not Thread-Safe	Thread-Safe
Faster Performance	Slow performance
one null key many null value	No null key and value
superclass : AbstractMap	Superclass : Dictionary

Name	Salary (Tk.)
Sakib Al Hasan	3,00,000
Mashrafe Bin Mortaza	null
Mushfiqur Rahim	2,50,000

Now write Java code for the following:

(i) Create an appropriate Hash-based collection to store the above information and then store the above information.

(ii) Increase Sakib Al Hasan's salary by 50,000 Tk.

```
HashMap<String, Integer> map
        = new HashMap<>();
map.put("Sakib Al Hasan", 300000);
map.put("Mashrafe Bin Mortaza", null);
map.put("Mushfiqur Rahim", 250000);
```

```
int salary= map.get("Sakib Al Hasan");
map.put("Sakib Al Hasan", salary + 500000);
```

(d) Consider the following Game of Thrones characters, their status, and the number of seasons active: **(7)**

Name	Status	NoOfSeasons
Jon Snow	Alive	7
Daenerys Targaryen	Alive	7
Ned Stark	Dead	1
Joffrey Baratheon	Dead	4
Tyrion Lannister	Alive	7

Write Java code for the following:

(i) Declare a single Enumeration named GOT for the above characters.

(ii) Find out the name of the character who was active for the least number of seasons (You can't simply write Ned Stark).

SECTION – B

```
i) enum GOT{
JonSnow("Jon Snow","Alive",7),
DaenerysTargaryen("Daenerys Targaryen","Alive",7),
NedStark("Ned Stark","Dead",1),
JoffreyBaratheon("Joffrey Baratheon","Alive",7),
TyrionLannisten("Tyrion Lannisten","Alive",7);
```

```
private int NoOfSeason;
private String name;
private String status;
// setter, getter, constructors
}
```

ii)

```
GOT got = GOT.JonSnow;
int least = got.getNoOfSeason();
for(GOT g : GOT.values()) {
 if ( g.getNoOfSeason()<least )
    {
      least = g.getNoOfSeason;
      got = g
    }
}
System.out.println(got.getName());
```

(c) A key/value pair stored in the vector "map". Write a genetic function that reads a key and then returns the associated value. **(10)**

```
public class Pair<K, V> {
    private final K key;
    private final V value;

    public Pair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public K getKey() {
        return key;
    }

    public V getValue() {
        return value;
    }

    @Override
    public String toString() {
        return "(" + key + ", " + value + ")";
    }
}
```

```
vector<Pair> vec = new Vector<>();
// element added
```

```
public <K,V> V genFunc (K key)
{
  for (Pair p : vec) {
  if (p.getkey() == key)
    return p.getValue();
```

```
    }
}
```

(c) Write a class Student that contains a HashMap with an integer as key and List of String as values. Add a method in the class to put 3 studentIds as the keys and corresponding [firstname, lastname] as values. Write another method to print all of them.

```
class Student {
private
 HashMap<Integer, List<String>>
                        map;
  Student() {
   map = new  HashMap<>();
    }
public  void  addHelper(int id, String
                    firstname,
                    String
                    secondname)
{
  List<String>  name = new ArrayList
                        <>();
  name.add(firstName);
```

```
    name.add(lastName);
    map.put(id, name);
  }
public void add3Student(সবগুলো Parameter)
  {
    addHelper(id1, fn1, Ln1);
    addHelper(id2, fn2, Ln2);
    addHelper(id3, fn3, Ln3);
  }

public void print()
  {
    int temp;
Iterator<Integer> it
        =map.keySet().Iterator();
while(it.hasNext())
```

```
{ temp = it.Next();
List<String> NameList
                =map.get(temp);

System.out.print(temp + ",  ");
 for(String S: NameList)
  {
  System.out.print( S +",  " );
 System.out.println();
   }
  }

}
```