1. In Java constructor is used to handle memory allocation efficiently. It initializes a new object from a class, with the same name, which resembles a method in java but does not have a return type. There are 3 types of constructors which are the default, no argument and parameterized constructors.
   a) Default Constructor
      A default constructor is when we do not implement any constructor in our class, so the java compiler will automatically create one, with an empty body, and will not be displayed in the source code. For example:

```
public Class Hello{
    public static void main(String[] args){
        Hello x = new Hello();
    }
}
```

      Once the source code above is compiled, it will look like this:

```
public Class Hello{
    Hello(){
    }
    public static void main(String[] args){
        Hello x = new Hello();
    }
}
```

   b) No Argument Constructor
      A no-argument constructor is when a constructor does not have any parameters and is very similar to a default constructor. However, different from a default constructor, there is a body that can have any code. For example:

```
public Class Hello{
    public Hello(){
        System.out.println("Hello World!");
    }
    public static void main(String[] args){
        new Hello();
    }
}
```

   c) Parameterized Constructor
      A parameterized constructor is a constructor with arguments or parameters. For example:

```
public Class Binusian{
    String id;
    String fullName;
    public Binusian(String id, String fullName){
        this.id = id;
        this.fullName = fullname;
    }
    public void printData(){
        System.out.println("ID: " + id);
        System.out.println("Full Name: " + fullName);
    }
    public static void main(String[] args){
        Binusian student1 = new Binusian(“B0001”,"Valerius Owen");
        Binusian staff1 = new Binusian(“B0002”,"Aldih Setiawan");
        student1.printData();
        staff1.printData();
    }
}
```

Destructor is a function in C++ to deallocate memory. However, since Java is a garbage collected language, it is a method in java that is automatically called which will free memory efficiently.

2. The concept of abstraction is the process of hiding unnecessary details from the user. In Java, it is done by making interfaces or abstract classes to allow the programmer to focus on the core. It can be also said that a driver does not need to know how the car accelerates or decelerates, all they need to know is to press the gas pedal to accelerate and the brake pedal to decelerate.

   Here is an example of the implementation of abstraction:

```
import java.util.*;
Import java.math.*;
import java.lang.reflect.Array;

public abstract class Binusian {
    private String id;
    private String name;

    public Binusian(String id, String name){
        this.id = id;
        setName(name);
    }

    public String getId() {
        return id;
    }

    public String getName() {
        return name;
    }

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

public class Student extends Binusian {

    public Student(String id, String name, String studentId, String major){
        super(id, name);
        this.studentId=studentId;
        setMajor(major);
    }

    private final String studentId;
    private String major;

    public String getStudentId() {
        return studentId;
    }

    public String getMajor() {
        return major;
    }

    public void setMajor(String major) {
        this.major = major;
    }
```

```
    public void study(){
        System.out.println( getName() + " is studying!");
    }
}

public interface Teacher {
    void teaching();
}

public class Lecturer extends Binusian implements Teacher {

    public Lecturer(String id, String name, String lecturerId, String salary) {
        super(id, name);
        this.lecturerId=lecturerId;
        setSalary(salary);
    }

    private final String lecturerId;
    private BigInteger salary;

    public String getLecturerId() {
        return lecturerId;
    }

    public BigInteger getSalary() {
        return salary;
    }

    public void setSalary(String salary) {
        this.salary = new BigInteger(salary);
    }

    @Override
    public void teaching(){
        System.out.println(getName() + " is teaching!");
    }
}

public class Instructor extends Student implements Teacher {

    public Instructor(String id, String name, String studentId, String major, String salary)
    {
        super(id, name, studentId, major);
        setSalary(salary);
    }

    private BigInteger salary;

    public BigInteger getSalary() {
        return salary;
    }

    public void setSalary(String salary) {
        this.salary = new BigInteger(salary);
    }

    @Override
    public void teaching(){
        System.out.println(getName() + " is instructing!");
```

```
    }

    @Override
    public void study(){
        System.out.println( getName() + " is not studying!");
    }
}


public class Main {
    public static void main(String[] args) {

        import model.*;

public class Main {
    public static void main(String[] args) {

        ArrayList<Binusian> binusianList = new ArrayList<>();

        Student student = new Student("BN2", "Owen","ST1","Computer Science");
        Binusian student2 = new Student("BN3", "Chandra", "ST2", "Computer Science");
        Lecturer lecturer1 = new Lecturer("BN4", "Budi", "D01", "10000000");
        Instructor instructor1 = new Instructor("BN5", "Siapa", "ST3", "Computer
         Science", "1500000");

        binusianList.add(student);
        binusianList.add(student2);
        binusianList.add(lecturer1);
        binusianList.add(instructor1);

        Student temp;
        if( student2 instanceof Student ){
            temp = (Student)student2;
            temp.getMajor();
        }
        System.out.println(student2 instanceof Student);
        lecturer1.teaching();
        ((Student)student2).study();
        student.study();

        for (Binusian b: binusianList){
            if( b instanceof Teacher ){
                ((Teacher) b).teaching();
            }
            if( b instanceof Student ){
                ((Student) b).study();
            }
        }
    }
}
```

3. Overloading is a method with different signatures where it has the same name however different parameters, whereas overriding is a method where a child class has a method that it inherits from it parent class but may change what it does. An example for overriding can be seen from the code above, there is a method called teaching, however there are two classes called Instructor and Lecturer with the same method Teaching from the class Teacher but does 2 different things where one prints out the name and “is teaching!” where another prints out the name and “is instructing!”. In comparison, overloading is when there are multiple functions with the same name but different parameters. Such as:

```
public int sum(int x, int y){
      return (x+y);
}

public int sum(int x, int y, int z){
      return(x+y+z);
}
```

4. Static method or variable is a property that is owned by a class and not an object, this allows different objects to share the same static property. A static variable is

```
class Counter{
    static int count=0;
    public void addOne(){
          count++;
    }
    public static void main(String args[]){
          Counter one = new Counter();
          Counter second = new Counter();
          for( int i = 0 ; i < 10 ; ++i ){
            one.addOne();
          }
          second.addOne();
          System.out.println("First counter is " + one.count);
          System.out.println("Second counter is " + second.count);
    }
}
```

5. Package in java is used to create differentiate domains and functionality. It can be used like the example from number 3 where there are multiple classes. A package named “models” can be made and filled with Teacher, Student, Lecturer and Instructor.

```
Import models.*;
```

Will import all in classes that are in the package “models”.