In a word document, you will need to briefly explain how you used these concepts into your program. You will need to provide examples from your code. Please be as brief as possible.

- Method overloading

Method overloading means we can create multiple methods of the same name in the same class, and all methods work in different ways, The method view_student_course() in the administrator class in the program is an example. They share the same method name, but take different input form and work in different ways.

```
public void view_student_course(String a, String b, studentDirectory c) {
        int count = 0;
        for (int i = 0; i < c.getDirectory().size(); i++) {
                if(c.getDirectory().get(i).getFirst_name().equals(a) &&
c.getDirectory().get(i).getLast_name().equals(b)) {
                        student d = c.getDirectory().get(i);
                        d.course_registered();
                        count ++;
                }
        }
        if (count ==0) {
                System.out.println("No such student found");
        }
}

public void view_student_course(int a, studentDirectory c) {
        int count = 0;
        for (int i = 0; i < c.getDirectory().size(); i++) {
                if(c.getDirectory().get(i).getStudent_id()==a) {
                        student d = c.getDirectory().get(i);
                        d.course_registered();
                        count ++;
                }
        }
        if (count ==0) {
                System.out.println("No such student found");
```

```
            }
        }
```

- Method overriding(at least two examples)

The method withdraw() in administrator class overrides the method in its parent class user.

```
public void withdraw (student a, String b, courseDirectory c) {
        for (int i =0; i <c.getDirectory().size(); i ++ ) {

        if(c.getDirectory().get(i).getCourse_id().equals(b)) {
                course drop = c.getDirectory().get(i);
                drop.remove_student(a);
                }
        }
}
```

The method register() in administrator class overrides the method in its parent class user.

```
public void register (student a, String b, courseDirectory c) {
        for (int i =0; i <c.getDirectory().size(); i ++ ) {
                if(c.getDirectory().get(i).getCourse_id().equals(b) && ! c.getDirectory().get(i).isfull()) {
                        course add_to_course = c.getDirectory().get(i);
                        add_to_course.add_student(a);
                }
        }
}
```

The method register() in student class overrides the method in its parent class user

```
public void register(course a) {
        course_registered.add(a);
```

```
        student b = new(student(this.first_name,this.last_name,this.student_id);
        a.add_student(b);
}
```

The method withdraw() in student class overrides the method in its parent class user

```
public void withdraw (course a) {
        course_registered.remove(a);
student b = new student(this.first_name,this.last_name,this.student_id);
        a.remove_student(b);
}
```

- Abstract class:

The User class is an abstract class. It contains many methods common to its subclasses, student class and administrator class. Two methods in this class are abstract that will be implemented in subclasses.

```
        public abstract void withdraw();

        public abstract void register();
```

- Inheritance
  The student class inherits from the user class.
  The administrator class inherits from the user class.
  The user class defined common attributes and methods shared by both student and administrator.

```
        public class student extends user implements stu {
        }

        public class administrator extends user implements Admin {
        }
```

- Polymorphism

The method with the name withdraw () has different forms in student class and the administrator class. The method takes different forms of input variables.

```
public void withdraw (student a, String b, courseDirectory c) {
        for (int i =0; i <c.getDirectory().size(); i ++ ) {
        if(c.getDirectory().get(i).getCourse_id().equals(b)) {
                course drop = c.getDirectory().get(i);
                drop.remove_student(a);
                }
        }
}

public void withdraw (course a) {
        course_registered.remove(a);
student b = new student(this.first_name,this.last_name,this.student_id);
        a.remove_student(b);
}
```

- Encapsulation
  In encapsulation, the data in a class is hidden from other classes which can be achieved by declaring all variables in the class as private and writing public methods in the class to set and get the values of variables. The course class in the program is an example.

```
public class course {

        private String course_name;
        private String course_id;
        private int course_capacity;
        private int current_student;
        private ArrayList <student> studentList = new ArrayList<student>();
        private String instructor;
        private int section;
        private String location;
```

```
// constructor
public course(){
}

public course(String a, String b, int c, int d,String f, int g, String h) {
		this.course_name = a;
		this.course_id = b;
		this.course_capacity = c;
		this.current_student = d;
		this.instructor = f;
		this.section = g;
		this.location = h;

}

// getters and setters

public String getCourse_name() {
		return course_name;
}

public void setCourse_name(String course_name) {
		this.course_name = course_name;
}

public String getCourse_id() {
		return course_id;
}

public void setCourse_id(String course_id) {
		this.course_id = course_id;
}

public int getCourse_capacity() {
		return course_capacity;
}
```

```
public void setCourse_capacity(int course_capacity) {
		this.course_capacity = course_capacity;
}

public int getCurrent_student() {
		return current_student;
}

public void setCurrent_student(int current_student) {
		this.current_student = current_student;
}



public String getInstructor() {
		return instructor;
}

public void setInstructor(String instructor) {
		this.instructor = instructor;
}

public int getSection() {
		return section;
}

public void setSection(int section) {
		this.section = section;
}

public String getLocation() {
		return location;
}

public void setLocation(String location) {
		this.location = location;
}
```

```
// this method checks if the course is full
public boolean isfull() {
	if (course_capacity == current_student) {
		return true;
	} else {
		return false;
	}
}

// this method adds student to the course
public void add_student(student a) {
	if(current_student + 1 > course_capacity) {
		System.out.println("Course is full");
	}else {
		studentList.add(a);
		current_student ++;
	}
}

// this method removes a student from the course
public void remove_student(student a) {
	studentList.remove(a);
	current_student --;
}

// this method display all students registered in the course
public void display_student() {
	for (int i = 0; i < studentList.size(); i++){
		studentList.get(i).student_info();
	}
}

public ArrayList<student> getStudentList() {
	return studentList;
}
```

```
	public void setStudentList(ArrayList<student> studentList) {
		this.studentList = studentList;
	}

	// this method display the information of this course
	public void printMe() {
		System.out.println("CourseName: "+ course_name);
		System.out.println("Courseid: "+ course_id);
		System.out.println("Number of student registered: "+current_student);
		System.out.println("Maximum number of students allowed to be registered"+ course_capacity);
		System.out.println("_____");
	}

}
```

- The concept of Abstract Data Types

An abstract data type is a special data type that is defined by a set of values and a set of operations on that type. The courseDirectory and studentDirectory in the program uses arraylist to store a set of courses objects and a set of student objects respectively.

```
private ArrayList<course> directory = new ArrayList<course>();

private ArrayList<student> directory = new ArrayList<student>();
```