

- Method overloading

```

94
95 public void viewAllCourses(ArrayList<Course> list) {
96     super.viewAllCourses(list);
97 }
98

127
128 }
129
130 public void viewAllCourses(String firstName, String lastName, ArrayList<Course> list) {
131     System.out.println("Courses registered by " + firstName + lastName + ": ");
132     for(int i=0; i<allStudent.size(); i++) {
133         if(allStudent.get(i).getFirstName().equalsIgnoreCase(firstName) && allStudent.get(i).getLastName().equalsIgnoreCase(lastName)) {
134             System.out.println(allStudent.get(i).getMyCourse());
135         }
136     }
137 }
138
139

```

In Class Admin, there are two Method viewAllCourses with the same name but different argument types.

- Method overriding (at least two examples)

```

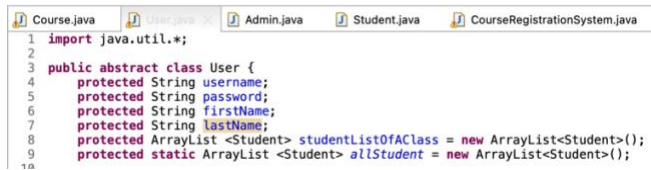
57 public void viewAllCourses(ArrayList<Course> list) {
58     for(int i=0; i<list.size(); i++) {
59         System.out.println(list.get(i).getCourseName() + ", " + list.get(i).getCourseId() + ", " + list.get(i).getMaximumStudents() + ", " +
60             list.get(i).getCourseInstructor() + ", " + list.get(i).getCourseSectionNumber() + ", " + list.get(i).getCourseLocation());
61     }
62 }
63
64 public void viewAllFullCourses(ArrayList<Course> list) {
65     int count = 0;
66     for(int i=0; i<list.size(); i++) {
67         if(list.get(i).getMaximumStudents() == list.get(i).getCurrentStudents()) {
68             System.out.println("All full courses are: " + list.get(i).getCourseName() + ", " + list.get(i).getCourseId() + ", " + list.get(i).getMaximumS
69                 + list.get(i).getCourseInstructor() + ", " + list.get(i).getCourseSectionNumber() + ", " + list.get(i).getCourseLocation());
70             count++;
71         }
72     }
73     if(count == 0)
74         System.out.println("There are no courses that are full.");
75 }
76
77
78

54
55 @Override
56 public void viewAllCourses(ArrayList<Course> list) {
57     for (int i=0; i<list.size(); i++) {
58         System.out.println("All courses: " + list.get(i).getCourseName());
59     }
60 }
61
62 @Override
63 public void viewAllFullCourses(ArrayList<Course> list) {
64     int count = 0;
65     for(int i=0; i<list.size(); i++) {
66         if(list.get(i).getMaximumStudents() != list.get(i).getCurrentStudents()) {
67             System.out.println("NOT full courses are: " + list.get(i).getCourseName() + ", " + list.get(i).getCourseId() + ", " + list.get(i).getMaximum
68                 + list.get(i).getCourseInstructor() + ", " + list.get(i).getCourseSectionNumber() + ", " + list.get(i).getCourseLocation());
69             count++;
70         }
71     }
72     if(count == 0)
73         System.out.println("All courses are full.");
74 }
75

```

Method viewAllCourses and Method viewAllFullCourses are overridden. In both examples, subclass (Student) overrides superclass (User), and only the subclass' method implementation is used.

- Abstract Class



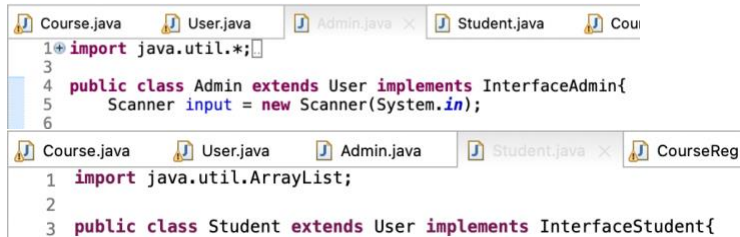
```

1 import java.util.*;
2
3 public abstract class User {
4     protected String username;
5     protected String password;
6     protected String firstName;
7     protected String lastName;
8     protected ArrayList<Student> studentListOfClass = new ArrayList<Student>();
9     protected static ArrayList<Student> allStudent = new ArrayList<Student>();
10 }

```

Class User is an abstract class.

- Inheritance



```

1 import java.util.*;
2
3 public class Admin extends User implements InterfaceAdmin {
4     Scanner input = new Scanner(System.in);
5 }

```

```

1 import java.util.ArrayList;
2
3 public class Student extends User implements InterfaceStudent {

```

Class Admin and Class Student extends Class User.

- Polymorphism



```

58 public void viewAllCourses(ArrayList<Course> list) {
59     for(int i=0; i<list.size(); i++) {
60         System.out.println(list.get(i).getCourseName() + ", " + list.get(i).getCourseId() + ", " + list.get(i).getMaximumStudents() + ", " +
61         + list.get(i).getCourseInstructor() + ", " + list.get(i).getCourseSectionNumber() + ", " + list.get(i).getCourseLocation());
62     }
63 }

```

```

64 public void viewAllFullCourses(ArrayList<Course> list) {
65     int count = 0;
66     for(int i=0; i<list.size(); i++) {
67         if(list.get(i).getMaximumStudents() == list.get(i).getCurrentStudents()) {
68             System.out.println("All full courses are: " + list.get(i).getCourseName() + ", " + list.get(i).getCourseId() + ", " + list.get(i).getMaximumS
69             + list.get(i).getCourseInstructor() + ", " + list.get(i).getCourseSectionNumber() + ", " + list.get(i).getCourseLocation());
70             count++;
71         }
72     }
73     if(count == 0)
74         System.out.println("There are no courses that are full.");
75 }

```

```

54
55 @Override
56 public void viewAllCourses(ArrayList<Course> list) {
57     for (int i=0; i<list.size(); i++) {
58         System.out.println("All courses: " + list.get(i).getCourseName());
59     }
60 }
61
62 @Override
63 public void viewAllFullCourses(ArrayList<Course> list) {
64     int count = 0;
65     for(int i=0; i<list.size(); i++) {
66         if(list.get(i).getMaximumStudents() != list.get(i).getCurrentStudents()) {
67             System.out.println("NOT full courses are: " + list.get(i).getCourseName() + ", " + list.get(i).getCourseId() + ", " + list.get(i).getMaximum
68             + list.get(i).getCourseInstructor() + ", " + list.get(i).getCourseSectionNumber() + ", " + list.get(i).getCourseLocation());
69             count++;
70         }
71     }
72     if(count == 0)
73         System.out.println("All courses are full.");
74 }
75 }

```

Method viewAllCourses and Method viewAllFullCourses are overridden. In both examples, subclass (Student) overrides superclass (User), and only the subclass' method implementation is used.

Method overriding is a form of polymorphism.

- Encapsulation

```
1 import java.util.*;
2
3 public class Course implements java.io.Serializable{
4     private String courseName;
5     private String courseId;
6     private int maximumStudents;
7     private int currentStudents;
8     private ArrayList<Student> listOfNames = new ArrayList<Student>();
9     private String courseInstructor;
10    private int courseSectionNumber;
11    private String courseLocation;
12 }
```

In Class Course, all the variables are declared private. Class Course interacts with other class through constructor, getters, setters, and other public method. This way, we can hide information and data.

- The concept of ADT (Abstract Data Types)

ADT is a mathematical description of a data structure. Each operation is described by an ADT, but not how it is carried out. There are many int, String, List (ArrayList) used in the code which are all ADT.

```
1 import java.util.*;
2
3 public class Course implements java.io.Serializable{
4     private String courseName;
5     private String courseId;
6     private int maximumStudents;
7     private int currentStudents;
8     private ArrayList<Student> listOfNames = new ArrayList<Student>();
9     private String courseInstructor;
10    private int courseSectionNumber;
11    private String courseLocation;
12 }
```