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**

The User Classcontains two constructors with different parameters.

Examples:

|  |
| --- |
| User(String username, String password) {   **this**.username = username;  **this**.password = password; } |

|  |
| --- |
| User(String username, String password, String firstName, String lastName) {   **this**.username = username;  **this**.password = password;  **this**.firstName = firstName;  **this**.lastName = lastName; } |

* **Method overriding (give at least two examples)**

Example 1:

The Admin Interface contains the following method without any implementation:

|  |
| --- |
| **public** **void** **viewAllCourseSections**(); |

The Admin Class overrides the method above:

|  |
| --- |
| **@Override** **public** **void** **viewAllCourseSections**() {   **for** (CourseSection courseSection : Data.getSingletonInstance().getCourseSectionArrayList()) {   System.out.println(courseSection);   } } |

Example 2:

The Student Interface contains the following method without any implementation:

|  |
| --- |
| **public** **void** **viewCoursesNotFull**(); |

The Student Class overrides the method above:

|  |
| --- |
| **@Override** **public** **void** **viewCoursesNotFull**() {  System.out.println("These are the courses that are not full: ");  **for** (CourseSection courseSection : Data.getSingletonInstance().getCourseSectionArrayList()) {  **if** (!courseSection.isFull()) {  System.out.println(courseSection.toStringForStudent());  }  }  System.out.println(); } |

* **Abstract Class**

Keeping User Class as an Abstract Class enables it to encapsulate the common attributes (variables and methods) of its child classes (Student and Admin) like any other class, but at the same time disallows it from instantiated.

|  |
| --- |
| **public** **abstract** **class** **User** **implements** **Serializable** {  **static** **final** **long** serialVersionUID = 4L;  **private** String username;  **private** String password;  **private** String firstName;  **private** String lastName;      User(String username, String password) {  **this**.username = username;  **this**.password = password;  }   User(String username, String password, String firstName, String lastName) {  **this**.username = username;  **this**.password = password;  **this**.firstName = firstName;  **this**.lastName = lastName;  }   *//Getter method for first name*  **public** String **getFirstName**() { **return** firstName; }  **public** String **getLastName**() { **return** lastName; }  **public** String **getPassword**() { **return** password; }  **public** String **getUsername**() { **return** username; } } |
|  |

* **Inheritance**

-The Student Class and Admin Class are Child classes of User Class. Thus, they inherit from the User Class.

Examples:

|  |
| --- |
| Student(String username, String password, String firstName, String lastName) {   **super**(username, password, firstName, lastName); }  **private** **Admin**(String username, String password) {  **super**(username, password); } |

The two examples above are invoking the constructors of their parent class(User Class)

* **Encapsulation**

|  |
| --- |
| **public** **class** **CourseSection** **implements** **Comparable**<**CourseSection**>, **Serializable** {  *//Class Instance Variables*  **private** String courseName;  **private** String courseId;  **private** **int** maxNumberStudent;  **private** String instructorName;  **private** **int** sectionNumber;  **private** String location;   *//Getter method for Course ID*    **public** String **getCourseId**()   **public** String **getLocation**()    *//Setter for Instructor name*  **public** **void** **setInstructorName**(String instructorName) |

The class Course Section is encapsulated because the class instance variables are declared as private and there are getter and setter methods that are used to access these variables. The methods are declared as public. In addition, the Course Section does not have direct accesses to the class variables. For example, courseSection.someVariable would not be allowed.

**▪ The concept of ADT (Abstract Data Types)**

|  |
| --- |
| **public** **interface** **StudentInterface** {   **public** **void** **register**(String courseName, **int** sectionNumber );   **public** **boolean** **withdraw**(String courseName, **int** sectionNumber);   **public** **void** **viewCoursesNotFull**();   **public** **void** **viewAllCourse**();   **public** String **getRegisteredClassString**();  } |

|  |
| --- |
| **public** **interface** **AdminInterface** {   *//Course Management*   **public** CourseSection **createCourseSection**(String courseName, String courseId, **int** maxStudent, String instructor, String location, **int** sectionNumber);   **public** **boolean** **deleteCourseSection**(String courseName, **int** sectionNumber);   **public** CourseSection **editCourseSection**(String courseName, **int** sectionNumber);   **public** **void** **displayCourseSectionInfo**(String courseId, **int** sectionNumber);   **public** Student **registerStudent**(String firstName, String lastName, String username, String password);    *//Reports*   **public** **void** **viewAllCourseSections**();   **public** **void** **viewFullCourseSection**();   **public** **void** **writeFullCourseSectionToFile**(String fileName) **throws** IOException;   **public** String **getStudentRegisteredCourseSectionsString**(String username);    **public** String **getStudentsInSpecificCourseSectionString**(String courseName, **int** sectionNumber);   **public** **void** **sortCourseSections**();  } |

The concept of ADT applies Student Interface and Admin Interface. These two interfaces only provide abstract methods without any implementations. The Student Class and Admin Class implement these two interfaces, respectively. The methods in these two interfaces do not necessarily show how they will be used in the program. The classes that implement the interfaces handle the details and processes.