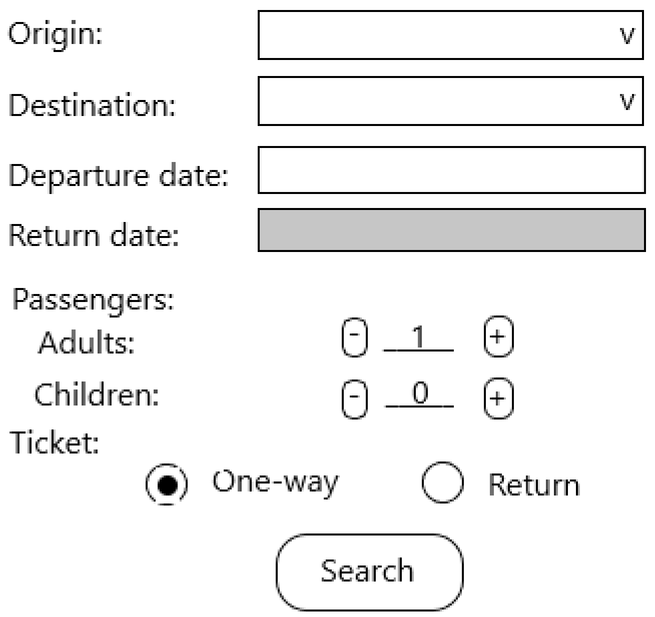
National University of Computer and Emerging Sciences, Lahore Campus

	Course Name:	**Software Construction & Development**	**Course Code:**	**CS-3001**
	Degree Program:	**BS(SE)**	**Semester:**	**Fall 2023**
	Exam Duration:	**60 Minutes**	**Total Marks:**	**45**
	Paper Date:	**10 - Nov - 2023**	**Weight:**	**15.00%**
	Section:	**ALL**	**Page(s):**	**6**
	Exam Type:	**Midterm-II**		

Student Name:_______________________________ **Roll No.**________________ **Section:**_____________

Instruction/Notes: Attempt all questions. Do not use pencil or red ink. In case of confusion or ambiguity make a reasonable assumption. **Do not attach any extra sheet**. Use extra sheet for rough work only. A **double-sided hand-written** cheat sheet is allowed but it shouldn't be photo-copied.

Question 1 [CLO-1] **5+10=15 points**



Consider a UI mock-up given above for a Flight Reservation System. **Origin** and **Destination** are dropdowns populated with a list of cities. **Departure date** and **Return date** are text boxes that stores date in **mm/dd/yyyy** format. **Passengers** information maintains the count of **adults** and **children** intending to travel. The count for each can be updated through corresponding **increment (+)** and **decrement (-)** buttons. **Ticket** type can be either **One-way** or **Return**, controlled through a radio button group. If **One-way**, then **Return date** is disabled, but if ticket type is selected as **Return** then **Return date** textbox will be enabled.

Write event handling code for:

(a) enable / disable **Return date** textbox based upon Ticket type.

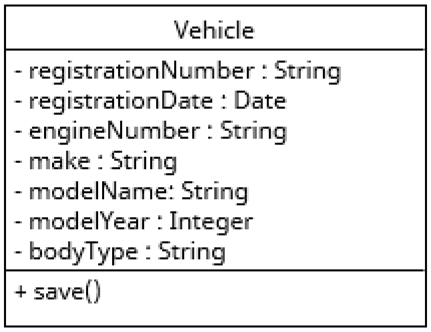
(b) increment / decrement count for adult and children passengers, <u>using a single event handler</u>.

Solution

```
class FlightReservationUI extends JFrame{

  // components go here ...

  public FlightReservationUI(){
     // instantiate components  and set layout and other properties ...
     TicketTypeListener ticketListener = new TicketTypeListener();
     PassengerCountListener countListener = new PassengerCountListener();
     onewayTicketRadioButton.addActionListener(ticketListener);
     returnTicketRadioButton.addActionListener(ticketListener);
     adultCountIncrementButton.addActionListener(countListener);
     adultCountDecrementButton.addActionListener(countListener);
     childrenCountIncrementButton.addActionListener(countListener);
     childrenCountDecrementButton.addActionListener(countListener);
  }

  private void increment(JTextField field){
    Integer currentValue = Integer.parseInteger(field.getText());
    field.setText("" + (currentValue + 1));
  }

  private void decrement(JTextField field){
    Integer currentValue = Integer.parseInteger(field.getText());
    field.setText("" + (currentValue - 1));
  }

  private class TicketTypeListener implements ActionListener{
    public void actionPerformed(ActionEvent e){
       if (e.getSource().equals(onewayTicketRadioButton){
         returnDateField.setEnabled(false);
       }

       if (e.getSource().equals(returnTicketRadioButton){
         returnDateField.setEnabled(true);
       }
    }
  }

  private class PassengerCountListener implements ActionListener{
    public void actionPerformed(ActionEvent e){
       if (e.getSource().equals(adultCountIncrementButton){
         increment(adultCountField);
       }

       if (e.getSource().equals(adultCountDecrementButton){
         decrement(adultCountField);
       }

       if (e.getSource().equals(childrenCountIncrementButton){
         increment(childrenCountField);
       }

       if (e.getSource().equals(childrenCountDecrementButton){
         decrement(childrenCountField);
       }
    }
  }
}
```

Question 2 [CLO-2] **15 points**

Excise Department maintains the registration record of vehicles using the following class:



Suppose they want to simultaneously store the information both in: (a) a relational database management system (e.g. MySql, SQL Server, etc.); and (b) a text file where each vehicle takes one line with its values comma-separated; for the purpose of reliability. Write code to support this requirement using relevant architectural and design patterns.

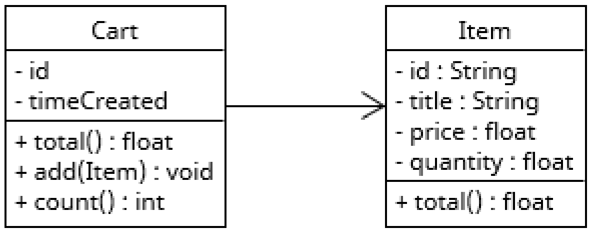
Solution

```
interface VehicleDAO{
  public boolean save(Hashtable<String,String> data);
}

class VehicleDbDAO implements VehicleDAO{
   public boolean save(Hashtable<String, String> data) {
        int count = 0;
        try{
            Connection conn = getConnection();   // assuming it is implemented
            PreparedStatement stmt = updateStatement(conn,data);
            count = stmt.executeUpdate();
            if (count == 0){
                stmt = insertStatement(conn,data);
                count = stmt.executeUpdate();
            }
        } catch(SQLException ex){
            return false;
        }
        return count > 0 ? true : false;
   }

    private PreparedStatement updateStatement(Connection conn,
                                  Hashtable<String,String> data) throws SQLException{
        String query = "update vehicle set regDate = ?, engNum = ?, make = ?, modelName
= ?, modelYear = ?, bodyType = ? where regNum = ?";
        PreparedStatement stmt = conn.prepareStatement(query);

        stmt.setString(1,data.get("registrationDate"));
        // other fields ...
        stmt.setString(7,data.get("registrationNumber"));

        return stmt;
    }
}
    // similarly write insertStatement function
```

```
public class VehicleFileDAO implements IVehicleDAO{

    File file;
    Hashtable<String,ArrayList<String>> contents;

    public VehicleFileDAO(String path){
        file = new File(path);
        contents = new Hashtable<>();
    }

    public boolean save(Hashtable<String, String> data) {
        ArrayList<String> row = new ArrayList<>();
        row.add(data.get("registrationNumber"));
        // similarly add other fields ...

        if(contents.get(data.get("registrationNumber")) != null){
            contents.replace(data.get("id"), row);
        } else{
            contents.put(data.get("registrationNumber"), row);
        }

        write();
        return true;
    }

    private void write(){
        try{
            BufferedWriter writer = new BufferedWriter(new FileWriter(file));
            for(ArrayList<String> row : contents.values()){
                for(String col : row){
                    writer.append(col + ",");
                }
                writer.append("\n");
            }
            writer.close();
        }
        catch(IOException ex){ }
    }
}

class Vehicle{
   private String registrationNumber;
   // other fields go here

   private ArrayList<IVehicleDAO> datasources;

   public Vehicle(){
     // instantiation and initialization
     datasources = new ArrayList<>();
     datasources.add(new VehicleDbDAO());
     datasources.add(new VehicleFileDAO());
   }

   public void save(){
     Hashtable<String,String> data = new Hashtable<>();
     data.put("registrationNumber",registrationNumber); // set other fields similarly

     for(IVehicleDAO datasource : datasources){
        datasource.save(data);
     }
   }
}
```

Question 3 [CLO-3] **10 points**

Write unit tests (using JUnit) for the following class diagram of a shopping cart system:



Item total provides the gross price (price x quantity) of a single item whereas Cart total provides the gross price of all items in a cart. Items can be added to the cart using add function. Count provides the number of items added to cart at any specific point in time.

Solution

```
class ItemTest{

  @Test
  public void testTotal(){
    Item i1 = new Item("i1","...",100,3);
    assertEquals(300,i1.total());

    Item i2 = new Item("i2","...",200,1);
    assertEquals(200,i2.total());

    Item i3 = new Item("i3","...",200,0);
    assertEquals(0,i3.total());
  }

}

class CartTest{

  Cart cart;

  public void setup(){
    cart = new Cart();
    Item i1 = new Item("i1","...",100,3);
    cart.add(i1);

    Item i2 = new Item("i2","...",200,1);
    cart.add(i2);

    Item i3 = new Item("i3","...",300,1);
    cart.add(i3);
  }

  @Test
  public void testCount(){
    assertEquals(3,cart.count());
  }

  @Test
  public void testAdd(){
    assertEquals(3,cart.count());
    Item i4 = new Item("i4","...",400,1);
    cart.add(i4);
    assertEquals(4,cart.count());

    Item i5 = new Item("i5","...",500,0);
    cart.add(i5);
    assertEquals(4,cart.count());
  }

  @Test
  public void testTotal(){
    if(cart.count() == 3){
      assertEquals(800,cart.total());
    }

    if(cart.count() == 4){
      assertEquals(1200,cart.total());
    }

  }

}
```