Domain Model:

Our task is to implement an interface ShiftMan to schedule workers into shifts, creating a roster for a shop. We can then get relevant information from the created roster such as to retrieve under/overstaffed shifts, shifts without managers, shifts where a staff member is the manager of, or the roster for a specified day or member. The result of these operations may be returned to the user. As such, error checking should be implemented.

To solve this issue, these concepts arise:

- *Time*
- *Days of the Week, Hours of a shift / working day*
- *Shop name*
- *Roster* (where you can set shifts, assign and register staff, retrieve roster info)
- *Manager or Worker*
- *Understaffed or overstaffed shifts, shifts with/without managers*
- *Shifts in general (perhaps have the details of each shift in a collection of lists)*
- *Staff employees* (where you can set details for each staff member, and list the shifts each staff is assigned to)
- *Error Checking Class* (where you can check for possible errors that may occur)

Possible relationships between concepts will be described.

- The **Roster** concept may relate to the **days of the week, hours of a shift/working day**, **staff employees**. Roster could be a collection, containing those concepts in lists.
- The **Shop** concept is also related to the **Roster** concept, as there are rosters placed for each shop to assign retail staff to serve customers.
- **Time / Days of the Week** relate to the **hours of a shift**. The working hours are set on a specified day at a certain time.

My chosen design relates to the domain model above as it includes many of the above concepts, with Roster being the main one, with collections of Shifts, Staff employees, Days of the week (in ArrayLists). I have an error checking class using checked exceptions (which turns out to be ShiftManServer), which checks for several errors (but not all). More detailed explanations of the chosen classes are as follows:

Colour coded: Classes in blue, Methods in green

Chosen Classes:

1) ShiftManServer - This class implements all the methods of ShiftMan, as required in the brief. It is mainly used for error checking using checked exceptions such as the ones named below. When newRoster() is called, it creates a Roster object, which ShiftManServer performs all its methods on. ShiftManServer's methods then pass the parameters to Roster which handles it.

i.e. ShiftManServer#registerStaff() checks if new roster has been created. If not, return error. If yes, it passes it into Roster#registerStaff() which then registers the staff, with the possibility of throwing an exception if staff name is duplicate or is empty. The exception is then caught by ShiftManServer, returning a String error message.

If an error occurs, several methods in Roster throw checked exceptions which is caught in ShiftManServer, producing an error message.

- **DuplicateStaffException** - Checks for duplicate staff being registered in registerStaff(). Duplicate means familyName and givenname are identical, case insensitive.
- **TimeException** - Checks if the starting and ending time given in parameters is valid in setWorkingHours() and addShift(). "Valid" means:
  - startTime != endTime
  - startTime must occur before endTime
- **RosterException** - Can do one of these:
  - In newRoster(), checks if shop name given is empty or null
  - In setWorkingHours() and addShift(), checks if day parameter is valid (matches one of the 7 days exactly) (Note: This can't be a TimeException because it's already used in those methods)
- **StaffException** - Checks if givenname or familyName in registerStaff() are empty or null
- **ShiftException** - Checks if shift in assignStaff() has been added in addShift()

Therefore, the requirement of dealing with at least *one* problem in at least *two* methods has been fulfilled. I have dealt with 1) invalid time issue and 2) invalid day in setWorkingHours() and addShift(), as well as others.

2) Roster - Contains the methods of ShiftManServer and is passed all parameters. Roster checks for errors and throws exceptions, to be caught by ShiftManServer. Has a list of Days (which then has a list of Shifts) and a list of StaffWorkers. These lists are iterated through when making shifts in addShift() to add a shift in a particular day, when checking for duplicate staff in registerStaff() and when retrieving lists in many of the methods e.g. getRosterForWorker() etc.

This class creates Day, DateTime, StaffWorker objects in newRoster(), setWorkingHours() and addShift(), and registerStaff() respectively.

Roster objects are created in ShiftManServer#newRoster(), and is used in ShiftManServer.

3) Day - Contains a list of Shifts that have been assigned for each day. Has an enum of DayOfWeek containing all 7 days of the week, with each day having an integer of its order. This helps with organising the list to be in chronological order in addShift(). Can add a Shift in its addShift() and retrieve existing shifts in getExistingShift().

This class creates Shift objects in addShift().

7 Day objects are created when Roster#newRoster() is called (which are stored in a List<Day> in Roster).

4) Shift - Contains a list of staff employees that are assigned to a specific Shift using addStaffMemberToList() The compareTo() method uses the enum of DayOfWeek to list all shifts in the order they occur (used in Day). Can assign a manager to a shift using assignManager().

This class creates DateTime objects in compareTo().

Shift objects are created in Day#addShift().

5) StaffWorker - encapsulates the staff employee's details, such as the Lists of Shifts they are assigned to, and assigned to as either worker or manager (three lists in total). The familyName of each staff is compared with another, which helps Roster arrange its staff list in alphabetical order.

StaffWorker objects are created in Roster#registerStaff() when a new staff is registered.

This class creates no objects.

6) DateTime - This class' functionality is for comparison of time. In the Roster class, objects of DateTime type are created in setWorkingHours() and addShift() to check if start/end times are valid. In the Shift class, DateTime objects are created which is used to compare and list shifts in chronological order. Provides an abstraction to hide time checking.

This class creates no objects.

Justification of Design:

With ShiftManServer being the error checking class, it passes all its methods to Roster. This makes it clear that ShiftManServer's purpose is to catch thrown exceptions from Roster.

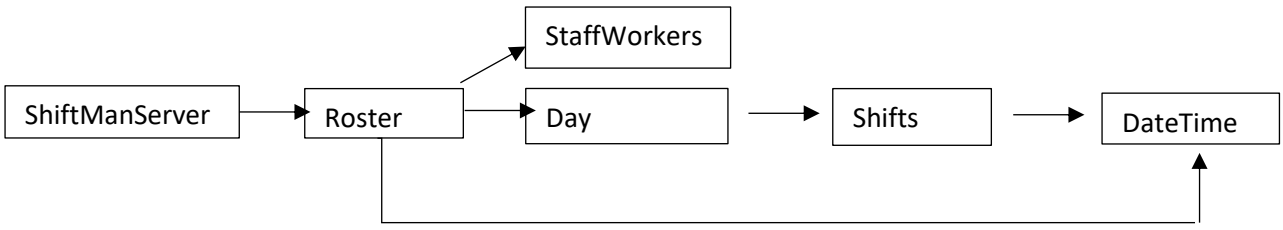
Roster is the "main" class which creates Day, StaffWorker and DateTime objects. Since many of the methods need to search for a specific day (i.e. when adding a shift for a day, or retrieving a list of rosters for a day) and need to search for an existing staff member (when checking for duplicates or retrieving roster for a staff member), there needed to be List of those Types ready to be iterated over. DateTime is created for checking validity of times.

In Day, shifts can be assigned to each of the working days. Therefore, shifts can be created here, and it contains a List of Shift objects which is needed to iterate over when the existing shift(s) is needed (i.e. for retrieving the roster/list of shifts for a day). This hides all the implementation which happens on each day, such as the shifts which occur.

In Shift, there must be a List of StaffWorkers to keep track of who is working on that shift object. This list is used to retrieve the workers for a specific shift in getRosterForDay() and to compare the List size to the minimum workers for understaffedShifts() and overstaffedShifts(). This hides all the information on each shift, such as which staff are working on that shift.

In StaffWorker, there needs to be a List of shifts that an employee is assigned to, and separate shifts where the employee is a worker or a manager. This is for easy access in the retrieval in shiftsManagedBy(), getRosterForWorker(), getUnassignedStaff(). This provides an abstraction to hide all the implementation of adding workers to shifts. Also encapsulates the details of each member, like their name, or the shifts they were assigned to.

The dependencies of the classes are as follows:



Object Counts: (more information in Appendix A)

In the custom **NewRosterDemo** I modified, I create a new roster, register three staff (two are valid) and add four shifts (two are valid). Therefore, 1 Roster, 7 Day, 2 StaffWorker and 2 Shift objects are created. 1 ShiftManServer object is also created (for checking for errors). I set the working hours 5 times (only 1 had a valid day/hours) and added 4 shifts (only 2 were valid). For all of them, DateTime objects were created to test their validity. Therefore 9 DateTime objects were created, but only 3 were kept (because they were valid times).

Then, I created a new roster - creating 1 Roster and 7 Day objects. All methods invoke using that new roster.

APPENDIX A: - NewRosterDemo vs Output

1) Executing methods without creating new roster: 1 ShiftManServer object is created.

```
System.out.println(">>Starting new roster demo");
ShiftMan scheduler = new ShiftManServer();
System.out.println(">>Setting working hours without having created new roster");
String status = scheduler.setWorkingHours("Monday", "09:00", "17:00"); //Should return an error
// Put status in {} so can tell when get empty string.
System.out.println("\tGot status {" + status + "}");
System.out.println("");

System.out.println(">>Adding a shift without having created new roster");
status = scheduler.addShift("Monday", "00:00", "23:59", "1"); //Should return an error
System.out.println("\tGot status {" + status + "}");
System.out.println("");

System.out.println(">>Register staff without having created new roster");
status = scheduler.registerStaff("Martin", "Tiangco"); //Should return an error
System.out.println("\tGot status {" + status + "}");
System.out.println("");

System.out.println(">>Assign shift to staff without having created new roster");
status = scheduler.assignStaff("Monday", "00:00", "23:59", "Martin", "Tiangco", false); //Should return an error
System.out.println("\tGot status {" + status + "}");
System.out.println("");

System.out.println(">>Get registered Staff without having created new roster");
List<String> status1 = scheduler.getRegisteredStaff(); //Should return an error
System.out.println("\tGot status {" + status1 + "}");
System.out.println("");

System.out.println(">>Get unassigned Staff without having created new roster");
status1 = scheduler.getUnassignedStaff(); //Should return an error
System.out.println("\tGot status {" + status1 + "}");
System.out.println("");

System.out.println(">>Get shifts without managers without having created new roster");
status1 = scheduler.shiftsWithoutManagers(); //Should return an error
System.out.println("\tGot status {" + status1 + "}");
System.out.println("");

System.out.println(">>Get roster for day without having created new roster");
status1 = scheduler.getRosterForDay("Monday"); //Should return an error
System.out.println("\tGot status {" + status1 + "}");
System.out.println("");
```

```
>>Starting new roster demo
>>Setting working hours without having created new roster
        Got status {%ERROR% --- Please create a new roster first.}

>>Adding a shift without having created new roster
        Got status {%ERROR% --- Please create a new roster first.}

>>Register staff without having created new roster
        Got status {%ERROR% --- Please create a new roster first.}

>>Assign shift to staff without having created new roster
        Got status {%ERROR% --- Please create a new roster first.}

>>Get registered Staff without having created new roster
        Got status {[ERROR: no roster has been created]}

>>Get unassigned Staff without having created new roster
        Got status {[ERROR: no roster has been created]}

>>Get shifts without managers without having created new roster
        Got status {[ERROR: no roster has been created]}

>>Get roster for day without having created new roster
        Got status {[ERROR: no roster has been created]}
```

2) Creating new roster:

In successful execution: 1 Roster and 7 Day objects created.

```
System.out.println(">>Create new roster with blank title -- should throw an exception");
status = scheduler.newRoster("");
System.out.println("\tGot status {" + status + "}");
System.out.println("");

System.out.println(">>Create new roster 'eScooters R Us' (status of {} means no error)");
status = scheduler.newRoster("eScooters R Us");
System.out.println("\tGot status {" + status + "}");
System.out.println("");
```

```
>>Create new roster with blank title -- should throw an exception
        Got status {%ERROR% --- Please provide a non-empty shop name.}

>>Create new roster 'eScooters R Us' (status of {} means no error)
        Got status {}
```

3) Setting working hours and testing DateTime:

5 DateTime objects are created since 5 working hour times are checked for validity.

```
System.out.println(">>Set working hours for TUEday to 09:00-17:00 (SHOULD RETURN ERROR MESSAGE SINCE INVALID DAY)");
status = scheduler.setWorkingHours("TUEday", "09:00", "17:00");
System.out.println("\tGot status {" + status + "}");
System.out.println("");

System.out.println(">>Set working hours for Monday to 00:00-24:00 (SHOULD RETURN ERROR MESSAGE SINCE INVALID TIME)");
status = scheduler.setWorkingHours("Monday", "00:00", "24:00");
System.out.println("\tGot status {" + status + "}");
System.out.println("");

System.out.println(">>Set working hours for Monday to 22:00-16:00 (ERROR SINCE endTime COMES BEFORE startTime)");
status = scheduler.setWorkingHours("Monday", "22:00", "16:00");
System.out.println("\tGot status {" + status + "}");
System.out.println("");

System.out.println(">>Set working hours for Monday to 22:00-22:00 (ERROR SINCE startTime = endTime)");
status = scheduler.setWorkingHours("Monday", "22:00", "22:00");
System.out.println("\tGot status {" + status + "}");
System.out.println("");

System.out.println(">>Set working hours for Tuesday to 09:00-17:00");
status = scheduler.setWorkingHours("Tuesday", "09:00", "17:00");
System.out.println("\tGot status {" + status + "}");
System.out.println("");
```

```
>>Set working hours for TUEday to 09:00-17:00 (SHOULD RETURN ERROR MESSAGE SINCE INVALID DAY)
        Got status {%ERROR% --- Please provide a valid name of day}

>>Set working hours for Monday to 00:00-24:00 (SHOULD RETURN ERROR MESSAGE SINCE INVALID TIME)
        Got status {%ERROR% --- Please provide a valid start and/or end time.}

>>Set working hours for Monday to 22:00-16:00 (ERROR SINCE endTime COMES BEFORE startTime)
        Got status {%ERROR% --- Please provide a valid start and/or end time.}

>>Set working hours for Monday to 22:00-22:00 (ERROR SINCE startTime = endTime)
        Got status {%ERROR% --- Please provide a valid start and/or end time.}

>>Set working hours for Tuesday to 09:00-17:00
        Got status {}
```

4) Registering Staff and getRegisteredStaff() and getUnassignedStaff()

I registered 2 non-duplicate staff members with “BAYTA Darell” being a duplicate of “Bayta Darell”. Therefore, 2 StaffWorker objects were created, not 3.

```
System.out.println(">>Register Bayta Darell as a staff member");
status = scheduler.registerStaff("Bayta", "Darell");
System.out.println("\tGot status {" + status + "}");
System.out.println("");

System.out.println(">>Register BAYTA DARELL as a staff member AGAIN (should return error)");
status = scheduler.registerStaff("BAYTA", "DARELL");
System.out.println("\tGot status {" + status + "}");
System.out.println("");

System.out.println(">>Register Hari Sheldon as a staff member");
status = scheduler.registerStaff("Hari", "Sheldon");
System.out.println("\tGot status {" + status + "}");
System.out.println("");

System.out.println(">>Get registered staff");
System.out.println(scheduler.getRegisteredStaff());
System.out.println("");

System.out.println(">>Display unassigned staff");
System.out.println(scheduler.getUnassignedStaff());
System.out.println("");
```

```
>>Register Bayta Darell as a staff member
        Got status {}

>>Register BAYTA DARELL as a staff member AGAIN (should return error)
        Got status {%ERROR% --- Staff already registered.}

>>Register Hari Sheldon as a staff member
        Got status {}

>>Get registered staff
[Bayta Darell, Hari Sheldon]

>>Display unassigned staff
[Bayta Darell, Hari Sheldon]
```

5) Adding shifts

For all 4 shifts, 4 DateTime objects were created to check their validity of day and start/end times. Only 2 were valid, and therefore 2 Shift objects were created.

```
System.out.println(">>Add shift 12:00-13:00 to WEDNESday (RETURNS ERROR SINCE INVALID DAY)");
status = scheduler.addShift("WEDNESday", "12:00", "13:00", "1");
System.out.println("\tGot status {" + status + "}");
System.out.println("");

System.out.println(">>Add shift 14:00-12:00 to Wednesday (RETURNS ERROR SINCE INVALID TIME)");
status = scheduler.addShift("Wednesday", "14:00", "12:00", "1");
System.out.println("\tGot status {" + status + "}");
System.out.println("");

System.out.println(">>Add shift 12:00-15:00 to Tuesday with minimum 2 workers");
status = scheduler.addShift("Tuesday", "12:00", "15:00", "2");
System.out.println("\tGot status {" + status + "}");
System.out.println("");

System.out.println(">>Add shift 09:00-12:00 to Tuesday with minimum 0 worker");
status = scheduler.addShift("Tuesday", "09:00", "12:00", "0");
System.out.println("\tGot status {" + status + "}");
System.out.println("");
```

```
>>Add shift 12:00-13:00 to WEDNESday (RETURNS ERROR SINCE INVALID DAY)
        Got status {%ERROR% --- Please provide a valid name of day}

>>Add shift 14:00-12:00 to Wednesday (RETURNS ERROR SINCE INVALID TIME)
        Got status {%ERROR% --- Please provide a valid start and/or end time.}

>>Add shift 12:00-15:00 to Tuesday with minimum 2 workers
        Got status {}

>>Add shift 09:00-12:00 to Tuesday with minimum 0 worker
        Got status {}
```

6) Assigning to shifts - No objects are created. All 3 assignments are valid.

```
System.out.println(">>Schedule Hari Sheldon as manager to Tuesday 12:00-15:00");
status = scheduler.assignStaff("Tuesday", "12:00", "15:00", "Hari", "Sheldon", true);
System.out.println("\tGot status {" + status +  "}");
System.out.println("");

System.out.println(">>Schedule Bayta Darell as worker to Tuesday 12:00-15:00");
status = scheduler.assignStaff("Tuesday", "12:00", "15:00", "Bayta", "Darell", false);
System.out.println("\tGot status {" + status + "}");
System.out.println("");

System.out.println(">>Schedule Hari Sheldon as worker to Tuesday 09:00-12:00");
status = scheduler.assignStaff("Tuesday", "09:00", "12:00", "Hari", "Sheldon", false);
System.out.println("\tGot status {" + status +  "}");
System.out.println("");
```

```
>>Schedule Hari Sheldon as manager to Tuesday 12:00-15:00
        Got status {}

>>Schedule Bayta Darell as worker to Tuesday 12:00-15:00
        Got status {}

>>Schedule Hari Sheldon as worker to Tuesday 09:00-12:00
        Got status {}
```

7) Getting roster for worker - 0 objects are created.

```
System.out.println(">>Display roster for staff Hari Sheldon");
System.out.println(scheduler.getRosterForWorker("Hari Sheldon"));
System.out.println("");
```

```
>>Display roster for staff Hari Sheldon
[Sheldon, Hari, Tuesday[09:00-12:00]]
```

8) Getting roster for day - 0 objects are created.

```
System.out.println(">>Display roster for Tuesday");
System.out.println(scheduler.getRosterForDay("Tuesday"));
System.out.println("");
```

```
>>Display roster for Tuesday
[eScooters R Us, Tuesday 09:00-17:00, Tuesday[09:00-12:00] [No manager assigned] [Hari Sheldon], Tuesday[12:00-15:00] Manager:Sheldon, Hari [Bayta Darell]]
```

9) Get shifts managed by or shifts without managers - 0 objects are created.

```
System.out.println(">>Display shifts managed by Hari Sheldon");
System.out.println(scheduler.getShiftsManagedBy("Hari Sheldon"));
System.out.println("");
```

```
System.out.println(">>Display shifts without managers");
System.out.println(scheduler.shiftsWithoutManagers());
System.out.println("");
```

```
>>Display shifts managed by Hari Sheldon
[Sheldon, Hari, Tuesday[12:00-15:00]]

>>Display shifts without managers
[Tuesday[09:00-12:00]]
```

10) Get under/overstaffed shifts - 0 objects are created.

```
System.out.println(">>Display understaffed shifts");
System.out.println(scheduler.understaffedShifts());
System.out.println("");

System.out.println(">>Display overstaffed shifts");
System.out.println(scheduler.overstaffedShifts());
System.out.println("");
```

```
>>Display understaffed shifts
[Tuesday[12:00-15:00]]

>>Display overstaffed shifts
[Tuesday[09:00-12:00]]
```

11) Creating a new roster and getting shifts for day - 1 new Roster object, and 7 Day objects are created.

```
System.out.println(">>Create a new Roster for 'Socks for Everyone'");
status = scheduler.newRoster("Socks for Everyone");
System.out.println("\tGot status {" + status + "}");

System.out.println(">>Display roster for Monday");
System.out.println(scheduler.getRosterForDay("Monday"));
System.out.println("");
System.out.println(">>End new roster demo");
```

```
>>Create a new Roster for 'Socks for Everyone'
        Got status {}
>>Display roster for Monday
[]

>>End new roster demo
```