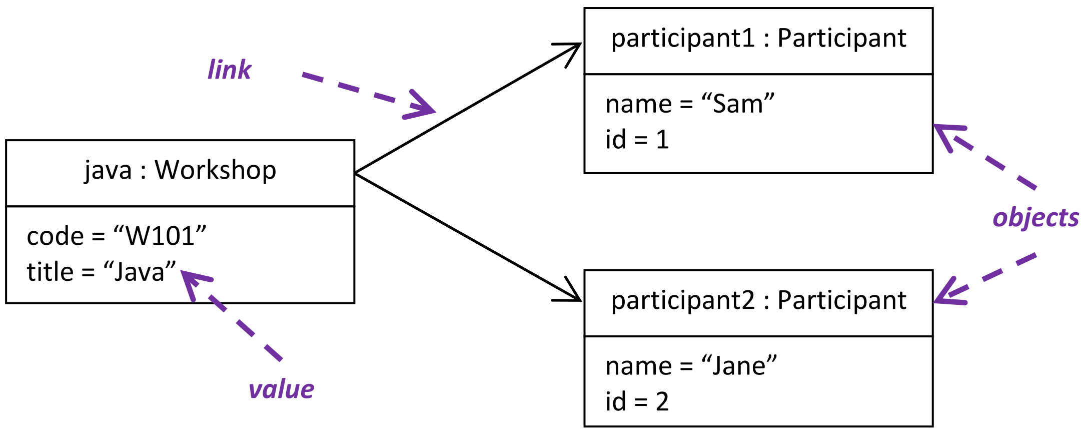
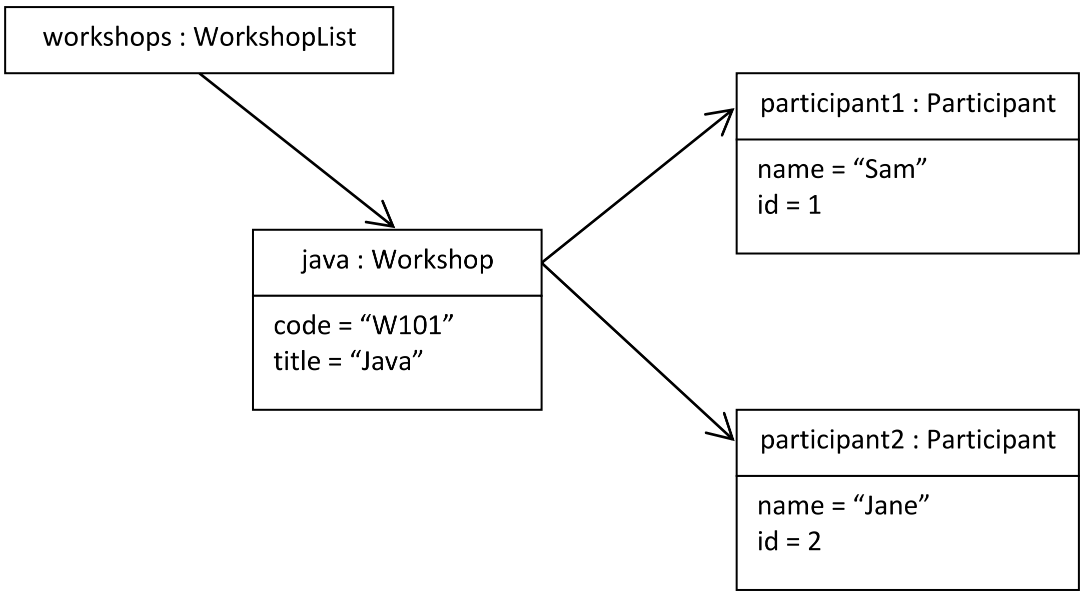
UECS2344 Software Design: Practical 5

1. The Object Diagram below shows the current state of objects in a system. It shows 3 objects with their values and the links between them. Based on the information you derive from the Object Diagram, draw a Class Diagram showing the classes (with attributes) and associations / relationships (with multiplicity and navigability).



2. The Object Diagram below shows the current state of objects in a system. The WorkshopList object keeps track of all Workshop objects in the system.



Redraw the Object Diagram after the following scenarios have occurred:

- Add new workshop (code: “W202”; title: “PHP”).
- Register participant Jane in the new workshop.

3. Consider the partial Code given below.
   (i) Draw a Design Class Diagram (showing only the details given, including multiplicity and navigability). Note: You need NOT include the App class.
   (ii) Draw an Object Diagram to show the objects and the links after each line is executed. Note: You need NOT show the values of the objects.
   (iii) Draw a Sequence Diagram.

(a) Uni-Directional To-One Association

```
public class Driver {

      // details left out

}

public class Bus {

      // some details left out

      private Driver theDriver;

      public void setDriver(Driver aDriver) {
            theDriver = aDriver;
      }

      public Driver getDriver {
            return theDriver;
      }
}

public class App {

      public static void main(String [] args) {

            Bus bus1 = new Bus();

            Driver driver1 = new Driver();

            bus1.setDriver(driver1);

            Driver theDriver = bus1.getDriver();

      }
}
```

(b) Bi-Directional One-To-One Association

```
public class Driver {

      // some details left out
      private Bus theBus;

      public void setBus(Bus aBus) {
            theBus = aBus;
      }

      public Bus getBus() {
            return theBus;
      }
}

public class Bus {

      // some details left out

      private Driver theDriver;

      public void setDriver(Driver aDriver) {
            theDriver = aDriver;
      }

      public Driver getDriver {
            return theDriver;
      }
}

public class App {

      public static void main(String [] args) {

            Bus bus1 = new Bus();

            Driver driver1 = new Driver();

            bus1.setDriver(driver1);

            driver1.setBus(bus1);

            Bus theBus = driver1.getBus();

      }
}
```

(c) Uni-Directional To-Many Association

```
public class Employee {

      // details left out
}

public class EmployeeList {

      private List<Employee> employees;

      public EmployeeList() {
            employees = new ArrayList<Employee>();
      }

      public void addEmployee(Employee anEMployee) {
            employees.add(anEmployee);
      }

}

public class App {

      public static void main(String [] args) {

            EmployeeList list = new EmployeeList();

            Employee e1 = new Employee();

            list.addEmployee(e1);

            Employee e2 = new Employee();

            list.addEmployee(e2);
      }
}
```

(d) Bi-Directional One-To-Many Association

```
public class Workshop {

      private List<Participant> participants;

      public Workshop() {
            participants = new ArrayList<Participant>();
      }

      public void enrollParticipant(Participant aParticipant) {
            participants.add(aParticipant);
      }
}

public class Participant {

      private Workshop theWorkshop;

      public Participant(Workshop theWorkshop) {
            this.theWorkshop = theWorkshop;
            theWorkshop.enrollParticipant(this);
      }
}


public class App {

      public static void main(String [] args) {

            Workshop workshop1 = new Workshop();

            Participant p1 = new Participant(workshop1);

            Participant p2 = new Participant(workshop1);

      }
}
```

(e) Bi-Directional Many-To-Many Association

```
public class Employee {

      private String name;

      private List<Job> jobsAssigned;

      public Employee(String name) {
            this.name = name;
            jobsAssigned = new ArrayList<Job>();
      }

      public String getName() {
            return name;
      }

      public void assignJob(Job job) {
            jobsAssigned.add(job);
            job.assignEmployee(this);
      }
}

public class Job {

      private String description;

      private List<Employee> employeesAssigned;

      public Job(String description) {
            this.description = description;
            employeesAssigned = new ArrayList<Employee>();
      }

      public String getDescription() {
            return description;
      }

      public void assignEmployee(Employee employee) {
            employeesAssigned.add(employee);
      }

}

public class App {

      public static void main(String [] args) {

            Employee sarah = new Employee("Sarah");

            Employee john = new Employee("John");

            Job job1 = new Job("Design");
            sarah.assignJob(job1);

            john.assignJob(job1);

            Job job2 = new Job("Testing");

            sarah.assignJob(job2);

      }
}
```

4. Refer to the code in Question 3 (e). Modify the code so that the list of jobs for an employee and the list of employees for a job can be displayed.

   Test your modified program will the following code (or similar code):

```
public class App {

      public static void main(String [] args) {

            Employee sarah = new Employee("Sarah");

            Employee john = new Employee("John");

            Job job1 = new Job("Design");

            sarah.assignJob(job1);

            john.assignJob(job1);

            Job job2 = new Job("Testing");

            sarah.assignJob(job2);

            System.out.println("---Sarah’s Jobs");
            List<Job> sarahJobs = sarah.getJobsAssigned();
            for (Job job : sarahJobs)
                  System.out.println(job.getDescription());


            System.out.println("---John’s Jobs");
            List<Job> johnJobs = john.getJobsAssigned();
            for (Job job : johnJobs)
                  System.out.println(job.getDescription());

            System.out.println("---Job1 Employees");
            List<Employee> job1Emps = job1.getEmployeesAssigned();
            for (Employee emp : job1Emps)
                  System.out.println(emp.getName());
            System.out.println("---Job2 Employees");
            List<Employee> job2Emps = job2.getEmployeesAssigned();
            for (Employee emp : job2Emps)
                  System.out.println(emp.getName());
      }
}
```

5. Currently, for the employee-job assignment, the assignJob() method is called on the Employee object, which calls the assignEmployee() method on the Job object. These two methods create the links between the objects. Modify the code so that we can call either assignJob() or assignEmployee() methods and the links between the objects are created correctly.

   Note: You must ensure that a job is assigned to an employee only once and an employee is assigned to a job only once. [*Hint: check if the job has already been added to the employee's list of jobs before adding and check if the employee has already been added to the job's list of employees before adding*.]

   Test your modified program will the following code (or similar code):

```
public class App {

      public static void main(String [] args) {

            Employee sarah = new Employee("Sarah");

            Job job1 = new Job("Design");

            sarah.assignJob(job1);
            job1.assignEmployee(sarah);

            System.out.println("---Sarah’s Jobs");
            List<Job> sarahJobs = sarah.getJobsAssigned();
            for (Job job : sarahJobs)
                  System.out.println(job.getDescription());

            System.out.println("---Job1 Employees");
            List<Employee> job1Emps = job1.getEmployeesAssigned();
            for (Employee emp : job1Emps)
                  System.out.println(emp.getName());
      }
}
```