```
0001. //%% NEW FILE Shift BEGINS HERE %%
0002.
0003. /*PLEASE DO NOT EDIT THIS CODE*/
0004. /*This code was generated using the UMPLE 1.29.1.4787.f023c4bb4 modeling
        language!*/
0005.
0006.
0007. import java.sql.Date;
0008.
0009. /**
0010.  * Working shifts that the employees have.
0011.  */
0012. // line 30 "model.ump"
0013. // line 112 "model.ump"
0014. public class Shift
0015. {
0016.
0017.   //------------------------
0018.   // MEMBER VARIABLES
0019.   //------------------------
0020.
0021.   //Shift Attributes
0022.   private Date date;
0023.   private int startTime;
0024.   private int endTime;
0025.
0026.   //Shift Associations
0027.   private Employee employee;
0028.
0029.   //------------------------
0030.   // CONSTRUCTOR
0031.   //------------------------
0032.
0033.   public Shift(Date aDate, int aStartTime, int aEndTime, Employee aEmployee)
0034.   {
0035.     date = aDate;
0036.     startTime = aStartTime;
0037.     endTime = aEndTime;
0038.     boolean didAddEmployee = setEmployee(aEmployee);
0039.     if (!didAddEmployee)
0040.     {
0041.       throw new RuntimeException("Unable to create shift due to employee. See
            http://manual.umple.org?RE002ViolationofAssociationMultiplicity.html");
0042.     }
0043.   }
0044.
0045.   //------------------------
0046.   // INTERFACE
0047.   //------------------------
0048.
0049.   public boolean setDate(Date aDate)
0050.   {
0051.     boolean wasSet = false;
0052.     date = aDate;
0053.     wasSet = true;
0054.     return wasSet;
0055.   }
0056.
0057.   public boolean setStartTime(int aStartTime)
0058.   {
```

```
0059.     boolean wasSet = false;
0060.     startTime = aStartTime;
0061.     wasSet = true;
0062.     return wasSet;
0063.   }
0064.
0065.   public boolean setEndTime(int aEndTime)
0066.   {
0067.     boolean wasSet = false;
0068.     endTime = aEndTime;
0069.     wasSet = true;
0070.     return wasSet;
0071.   }
0072.
0073.   public Date getDate()
0074.   {
0075.     return date;
0076.   }
0077.
0078.   public int getStartTime()
0079.   {
0080.     return startTime;
0081.   }
0082.
0083.   public int getEndTime()
0084.   {
0085.     return endTime;
0086.   }
0087.   /* Code from template association_GetOne */
0088.   public Employee getEmployee()
0089.   {
0090.     return employee;
0091.   }
0092.   /* Code from template association_SetOneToMandatoryMany */
0093.   public boolean setEmployee(Employee aEmployee)
0094.   {
0095.     boolean wasSet = false;
0096.     //Must provide employee to shift
0097.     if (aEmployee == null)
0098.     {
0099.       return wasSet;
0100.     }
0101.
0102.     if (employee != null && employee.numberOfShifts() <=
            Employee.minimumNumberOfShifts())
0103.     {
0104.       return wasSet;
0105.     }
0106.
0107.     Employee existingEmployee = employee;
0108.     employee = aEmployee;
0109.     if (existingEmployee != null && !existingEmployee.equals(aEmployee))
0110.     {
0111.       boolean didRemove = existingEmployee.removeShift(this);
0112.       if (!didRemove)
0113.       {
0114.         employee = existingEmployee;
0115.         return wasSet;
0116.       }
0117.     }
0118.     employee.addShift(this);
```

```
0119.     wasSet = true;
0120.     return wasSet;
0121.   }
0122.
0123.   public void delete()
0124.   {
0125.     Employee placeholderEmployee = employee;
0126.     this.employee = null;
0127.     if(placeholderEmployee != null)
0128.     {
0129.       placeholderEmployee.removeShift(this);
0130.     }
0131.   }
0132.
0133.
0134.   public String toString()
0135.   {
0136.     return super.toString() + "["+
0137.            "startTime" + ":" + getStartTime()+ "," +
0138.            "endTime" + ":" + getEndTime()+ "]" +
                   System.getProperties().getProperty("line.separator") +
0139.            "  " + "date" + "=" + (getDate() != null ? !getDate().equals(this)  ?
                   getDate().toString().replaceAll("  ","    ") : "this" : "null") +
                   System.getProperties().getProperty("line.separator") +
0140.            "  " + "employee = "+(getEmployee()!=null?
                   Integer.toHexString(System.identityHashCode(getEmployee())):"null");
0141.   }
0142. }
0143.
0144.
0145.
0146. //%% NEW FILE Employee BEGINS HERE %%
0147.
0148. /*PLEASE DO NOT EDIT THIS CODE*/
0149. /*This code was generated using the UMPLE 1.29.1.4787.f023c4bb4 modeling
        language!*/
0150.
0151.
0152. import java.util.*;
0153. import java.sql.Date;
0154.
0155. /**
0156.  * The people who work at the hospital.
0157.  */
0158. // line 20 "model.ump"
0159. // line 122 "model.ump"
0160. public class Employee
0161. {
0162.
0163.   //------------------------
0164.   // MEMBER VARIABLES
0165.   //------------------------
0166.
0167.   //Employee Attributes
0168.   private String name;
0169.   private int id;
0170.   private int salary;
0171.
0172.   //Employee Associations
0173.   private List<Shift> shifts;
0174.   private List<Privilege> privileges;
```

```
0175.   private Hospital hospital;
0176.   private List<Ward> wards;
0177.
0178.   //------------------------
0179.   // CONSTRUCTOR
0180.   //------------------------
0181.
0182.   public Employee(String aName, int aId, int aSalary, Hospital aHospital,
           Ward... allWards)
0183.   {
0184.     name = aName;
0185.     id = aId;
0186.     salary = aSalary;
0187.     shifts = new ArrayList<Shift>();
0188.     privileges = new ArrayList<Privilege>();
0189.     boolean didAddHospital = setHospital(aHospital);
0190.     if (!didAddHospital)
0191.     {
0192.       throw new RuntimeException("Unable to create employee due to hospital. See
             http://manual.umple.org?RE002ViolationofAssociationMultiplicity.html");
0193.     }
0194.     wards = new ArrayList<Ward>();
0195.     boolean didAddWards = setWards(allWards);
0196.     if (!didAddWards)
0197.     {
0198.       throw new RuntimeException("Unable to create Employee, must have at least 1
             wards. See http://manual.umple.org?
             RE002ViolationofAssociationMultiplicity.html");
0199.     }
0200.   }
0201.
0202.   //------------------------
0203.   // INTERFACE
0204.   //------------------------
0205.
0206.   public boolean setName(String aName)
0207.   {
0208.     boolean wasSet = false;
0209.     name = aName;
0210.     wasSet = true;
0211.     return wasSet;
0212.   }
0213.
0214.   public boolean setId(int aId)
0215.   {
0216.     boolean wasSet = false;
0217.     id = aId;
0218.     wasSet = true;
0219.     return wasSet;
0220.   }
0221.
0222.   public boolean setSalary(int aSalary)
0223.   {
0224.     boolean wasSet = false;
0225.     salary = aSalary;
0226.     wasSet = true;
0227.     return wasSet;
0228.   }
0229.
0230.   public String getName()
0231.   {
```

```
0232.     return name;
0233.   }
0234.
0235.   public int getId()
0236.   {
0237.     return id;
0238.   }
0239.
0240.   public int getSalary()
0241.   {
0242.     return salary;
0243.   }
0244.   /* Code from template association_GetMany */
0245.   public Shift getShift(int index)
0246.   {
0247.     Shift aShift = shifts.get(index);
0248.     return aShift;
0249.   }
0250.
0251.   public List<Shift> getShifts()
0252.   {
0253.     List<Shift> newShifts = Collections.unmodifiableList(shifts);
0254.     return newShifts;
0255.   }
0256.
0257.   public int numberOfShifts()
0258.   {
0259.     int number = shifts.size();
0260.     return number;
0261.   }
0262.
0263.   public boolean hasShifts()
0264.   {
0265.     boolean has = shifts.size() > 0;
0266.     return has;
0267.   }
0268.
0269.   public int indexOfShift(Shift aShift)
0270.   {
0271.     int index = shifts.indexOf(aShift);
0272.     return index;
0273.   }
0274.   /* Code from template association_GetMany */
0275.   public Privilege getPrivilege(int index)
0276.   {
0277.     Privilege aPrivilege = privileges.get(index);
0278.     return aPrivilege;
0279.   }
0280.
0281.   public List<Privilege> getPrivileges()
0282.   {
0283.     List<Privilege> newPrivileges = Collections.unmodifiableList(privileges);
0284.     return newPrivileges;
0285.   }
0286.
0287.   public int numberOfPrivileges()
0288.   {
0289.     int number = privileges.size();
0290.     return number;
0291.   }
0292.
```

```
0293.   public boolean hasPrivileges()
0294.   {
0295.     boolean has = privileges.size() > 0;
0296.     return has;
0297.   }
0298.
0299.   public int indexOfPrivilege(Privilege aPrivilege)
0300.   {
0301.     int index = privileges.indexOf(aPrivilege);
0302.     return index;
0303.   }
0304.   /* Code from template association_GetOne */
0305.   public Hospital getHospital()
0306.   {
0307.     return hospital;
0308.   }
0309.   /* Code from template association_GetMany */
0310.   public Ward getWard(int index)
0311.   {
0312.     Ward aWard = wards.get(index);
0313.     return aWard;
0314.   }
0315.
0316.   public List<Ward> getWards()
0317.   {
0318.     List<Ward> newWards = Collections.unmodifiableList(wards);
0319.     return newWards;
0320.   }
0321.
0322.   public int numberOfWards()
0323.   {
0324.     int number = wards.size();
0325.     return number;
0326.   }
0327.
0328.   public boolean hasWards()
0329.   {
0330.     boolean has = wards.size() > 0;
0331.     return has;
0332.   }
0333.
0334.   public int indexOfWard(Ward aWard)
0335.   {
0336.     int index = wards.indexOf(aWard);
0337.     return index;
0338.   }
0339.   /* Code from template association_IsNumberOfValidMethod */
0340.   public boolean isNumberOfShiftsValid()
0341.   {
0342.     boolean isValid = numberOfShifts() >= minimumNumberOfShifts();
0343.     return isValid;
0344.   }
0345.   /* Code from template association_MinimumNumberOfMethod */
0346.   public static int minimumNumberOfShifts()
0347.   {
0348.     return 1;
0349.   }
0350.   /* Code from template association_AddMandatoryManyToOne */
0351.   public Shift addShift(Date aDate, int aStartTime, int aEndTime)
0352.   {
0353.     Shift aNewShift = new Shift(aDate, aStartTime, aEndTime, this);
```

```
0354.     return aNewShift;
0355.   }
0356.
0357.   public boolean addShift(Shift aShift)
0358.   {
0359.     boolean wasAdded = false;
0360.     if (shifts.contains(aShift)) { return false; }
0361.     Employee existingEmployee = aShift.getEmployee();
0362.     boolean isNewEmployee = existingEmployee != null &&
            !this.equals(existingEmployee);
0363.
0364.     if (isNewEmployee && existingEmployee.numberOfShifts() <=
            minimumNumberOfShifts())
0365.     {
0366.       return wasAdded;
0367.     }
0368.     if (isNewEmployee)
0369.     {
0370.       aShift.setEmployee(this);
0371.     }
0372.     else
0373.     {
0374.       shifts.add(aShift);
0375.     }
0376.     wasAdded = true;
0377.     return wasAdded;
0378.   }
0379.
0380.   public boolean removeShift(Shift aShift)
0381.   {
0382.     boolean wasRemoved = false;
0383.     //Unable to remove aShift, as it must always have a employee
0384.     if (this.equals(aShift.getEmployee()))
0385.     {
0386.       return wasRemoved;
0387.     }
0388.
0389.     //employee already at minimum (1)
0390.     if (numberOfShifts() <= minimumNumberOfShifts())
0391.     {
0392.       return wasRemoved;
0393.     }
0394.
0395.     shifts.remove(aShift);
0396.     wasRemoved = true;
0397.     return wasRemoved;
0398.   }
0399.   /* Code from template association_AddIndexControlFunctions */
0400.   public boolean addShiftAt(Shift aShift, int index)
0401.   {
0402.     boolean wasAdded = false;
0403.     if(addShift(aShift))
0404.     {
0405.       if(index < 0 ) { index = 0; }
0406.       if(index > numberOfShifts()) { index = numberOfShifts() - 1; }
0407.       shifts.remove(aShift);
0408.       shifts.add(index, aShift);
0409.       wasAdded = true;
0410.     }
0411.     return wasAdded;
0412.   }
```

```
0413.
0414.   public boolean addOrMoveShiftAt(Shift aShift, int index)
0415.   {
0416.     boolean wasAdded = false;
0417.     if(shifts.contains(aShift))
0418.     {
0419.       if(index < 0 ) { index = 0; }
0420.       if(index > numberOfShifts()) { index = numberOfShifts() - 1; }
0421.       shifts.remove(aShift);
0422.       shifts.add(index, aShift);
0423.       wasAdded = true;
0424.     }
0425.     else
0426.     {
0427.       wasAdded = addShiftAt(aShift, index);
0428.     }
0429.     return wasAdded;
0430.   }
0431.   /* Code from template association_IsNumberOfValidMethod */
0432.   public boolean isNumberOfPrivilegesValid()
0433.   {
0434.     boolean isValid = numberOfPrivileges() >= minimumNumberOfPrivileges();
0435.     return isValid;
0436.   }
0437.   /* Code from template association_MinimumNumberOfMethod */
0438.   public static int minimumNumberOfPrivileges()
0439.   {
0440.     return 1;
0441.   }
0442.   /* Code from template association_AddMandatoryManyToOne */
0443.   public Privilege addPrivilege(String aPrivilege)
0444.   {
0445.     Privilege aNewPrivilege = new Privilege(aPrivilege, this);
0446.     return aNewPrivilege;
0447.   }
0448.
0449.   public boolean addPrivilege(Privilege aPrivilege)
0450.   {
0451.     boolean wasAdded = false;
0452.     if (privileges.contains(aPrivilege)) { return false; }
0453.     Employee existingEmployee = aPrivilege.getEmployee();
0454.     boolean isNewEmployee = existingEmployee != null &&
          !this.equals(existingEmployee);
0455.
0456.     if (isNewEmployee && existingEmployee.numberOfPrivileges() <=
          minimumNumberOfPrivileges())
0457.     {
0458.       return wasAdded;
0459.     }
0460.     if (isNewEmployee)
0461.     {
0462.       aPrivilege.setEmployee(this);
0463.     }
0464.     else
0465.     {
0466.       privileges.add(aPrivilege);
0467.     }
0468.     wasAdded = true;
0469.     return wasAdded;
0470.   }
0471.
```

```
0472.   public boolean removePrivilege(Privilege aPrivilege)
0473.   {
0474.     boolean wasRemoved = false;
0475.     //Unable to remove aPrivilege, as it must always have a employee
0476.     if (this.equals(aPrivilege.getEmployee()))
0477.     {
0478.       return wasRemoved;
0479.     }
0480.
0481.     //employee already at minimum (1)
0482.     if (numberOfPrivileges() <= minimumNumberOfPrivileges())
0483.     {
0484.       return wasRemoved;
0485.     }
0486.
0487.     privileges.remove(aPrivilege);
0488.     wasRemoved = true;
0489.     return wasRemoved;
0490.   }
0491.   /* Code from template association_AddIndexControlFunctions */
0492.   public boolean addPrivilegeAt(Privilege aPrivilege, int index)
0493.   {
0494.     boolean wasAdded = false;
0495.     if(addPrivilege(aPrivilege))
0496.     {
0497.       if(index < 0 ) { index = 0; }
0498.       if(index > numberOfPrivileges()) { index = numberOfPrivileges() - 1; }
0499.       privileges.remove(aPrivilege);
0500.       privileges.add(index, aPrivilege);
0501.       wasAdded = true;
0502.     }
0503.     return wasAdded;
0504.   }
0505.
0506.   public boolean addOrMovePrivilegeAt(Privilege aPrivilege, int index)
0507.   {
0508.     boolean wasAdded = false;
0509.     if(privileges.contains(aPrivilege))
0510.     {
0511.       if(index < 0 ) { index = 0; }
0512.       if(index > numberOfPrivileges()) { index = numberOfPrivileges() - 1; }
0513.       privileges.remove(aPrivilege);
0514.       privileges.add(index, aPrivilege);
0515.       wasAdded = true;
0516.     }
0517.     else
0518.     {
0519.       wasAdded = addPrivilegeAt(aPrivilege, index);
0520.     }
0521.     return wasAdded;
0522.   }
0523.   /* Code from template association_SetOneToMany */
0524.   public boolean setHospital(Hospital aHospital)
0525.   {
0526.     boolean wasSet = false;
0527.     if (aHospital == null)
0528.     {
0529.       return wasSet;
0530.     }
0531.
0532.     Hospital existingHospital = hospital;
```

```
0533.     hospital = aHospital;
0534.     if (existingHospital != null && !existingHospital.equals(aHospital))
0535.     {
0536.       existingHospital.removeEmployee(this);
0537.     }
0538.     hospital.addEmployee(this);
0539.     wasSet = true;
0540.     return wasSet;
0541.   }
0542.   /* Code from template association_IsNumberOfValidMethod */
0543.   public boolean isNumberOfWardsValid()
0544.   {
0545.     boolean isValid = numberOfWards() >= minimumNumberOfWards();
0546.     return isValid;
0547.   }
0548.   /* Code from template association_MinimumNumberOfMethod */
0549.   public static int minimumNumberOfWards()
0550.   {
0551.     return 1;
0552.   }
0553.   /* Code from template association_AddManyToManyMethod */
0554.   public boolean addWard(Ward aWard)
0555.   {
0556.     boolean wasAdded = false;
0557.     if (wards.contains(aWard)) { return false; }
0558.     wards.add(aWard);
0559.     if (aWard.indexOfEmployee(this) != -1)
0560.     {
0561.       wasAdded = true;
0562.     }
0563.     else
0564.     {
0565.       wasAdded = aWard.addEmployee(this);
0566.       if (!wasAdded)
0567.       {
0568.         wards.remove(aWard);
0569.       }
0570.     }
0571.     return wasAdded;
0572.   }
0573.   /* Code from template association_AddMStarToMany */
0574.   public boolean removeWard(Ward aWard)
0575.   {
0576.     boolean wasRemoved = false;
0577.     if (!wards.contains(aWard))
0578.     {
0579.       return wasRemoved;
0580.     }
0581.
0582.     if (numberOfWards() <= minimumNumberOfWards())
0583.     {
0584.       return wasRemoved;
0585.     }
0586.
0587.     int oldIndex = wards.indexOf(aWard);
0588.     wards.remove(oldIndex);
0589.     if (aWard.indexOfEmployee(this) == -1)
0590.     {
0591.       wasRemoved = true;
0592.     }
0593.     else
```

```
0594.     {
0595.       wasRemoved = aWard.removeEmployee(this);
0596.       if (!wasRemoved)
0597.       {
0598.         wards.add(oldIndex,aWard);
0599.       }
0600.     }
0601.     return wasRemoved;
0602.   }
0603.   /* Code from template association_SetMStarToMany */
0604.   public boolean setWards(Ward... newWards)
0605.   {
0606.     boolean wasSet = false;
0607.     ArrayList<Ward> verifiedWards = new ArrayList<Ward>();
0608.     for (Ward aWard : newWards)
0609.     {
0610.       if (verifiedWards.contains(aWard))
0611.       {
0612.         continue;
0613.       }
0614.       verifiedWards.add(aWard);
0615.     }
0616.
0617.     if (verifiedWards.size() != newWards.length || verifiedWards.size() <
          minimumNumberOfWards())
0618.     {
0619.       return wasSet;
0620.     }
0621.
0622.     ArrayList<Ward> oldWards = new ArrayList<Ward>(wards);
0623.     wards.clear();
0624.     for (Ward aNewWard : verifiedWards)
0625.     {
0626.       wards.add(aNewWard);
0627.       if (oldWards.contains(aNewWard))
0628.       {
0629.         oldWards.remove(aNewWard);
0630.       }
0631.       else
0632.       {
0633.         aNewWard.addEmployee(this);
0634.       }
0635.     }
0636.
0637.     for (Ward anOldWard : oldWards)
0638.     {
0639.       anOldWard.removeEmployee(this);
0640.     }
0641.     wasSet = true;
0642.     return wasSet;
0643.   }
0644.   /* Code from template association_AddIndexControlFunctions */
0645.   public boolean addWardAt(Ward aWard, int index)
0646.   {
0647.     boolean wasAdded = false;
0648.     if(addWard(aWard))
0649.     {
0650.       if(index < 0 ) { index = 0; }
0651.       if(index > numberOfWards()) { index = numberOfWards() - 1; }
0652.       wards.remove(aWard);
0653.       wards.add(index, aWard);
```

```
0654.       wasAdded = true;
0655.     }
0656.     return wasAdded;
0657.   }
0658.
0659.   public boolean addOrMoveWardAt(Ward aWard, int index)
0660.   {
0661.     boolean wasAdded = false;
0662.     if(wards.contains(aWard))
0663.     {
0664.       if(index < 0 ) { index = 0; }
0665.       if(index > numberOfWards()) { index = numberOfWards() - 1; }
0666.       wards.remove(aWard);
0667.       wards.add(index, aWard);
0668.       wasAdded = true;
0669.     }
0670.     else
0671.     {
0672.       wasAdded = addWardAt(aWard, index);
0673.     }
0674.     return wasAdded;
0675.   }
0676.
0677.   public void delete()
0678.   {
0679.     for(int i=shifts.size(); i > 0; i--)
0680.     {
0681.       Shift aShift = shifts.get(i - 1);
0682.       aShift.delete();
0683.     }
0684.     for(int i=privileges.size(); i > 0; i--)
0685.     {
0686.       Privilege aPrivilege = privileges.get(i - 1);
0687.       aPrivilege.delete();
0688.     }
0689.     Hospital placeholderHospital = hospital;
0690.     this.hospital = null;
0691.     if(placeholderHospital != null)
0692.     {
0693.       placeholderHospital.removeEmployee(this);
0694.     }
0695.     ArrayList<Ward> copyOfWards = new ArrayList<Ward>(wards);
0696.     wards.clear();
0697.     for(Ward aWard : copyOfWards)
0698.     {
0699.       aWard.removeEmployee(this);
0700.     }
0701.   }
0702.
0703.
0704.   public String toString()
0705.   {
0706.     return super.toString() + "["+
0707.            "name" + ":" + getName()+ "," +
0708.            "id" + ":" + getId()+ "," +
0709.            "salary" + ":" + getSalary()+ "]" +
                 System.getProperties().getProperty("line.separator") +
0710.            "  " + "hospital = "+(getHospital()!=null?
                 Integer.toHexString(System.identityHashCode(getHospital())):"null");
0711.   }
0712. }
```

```
0713.
0714.
0715.
0716. //%% NEW FILE Ward BEGINS HERE %%
0717.
0718. /*PLEASE DO NOT EDIT THIS CODE*/
0719. /*This code was generated using the UMPLE 1.29.1.4787.f023c4bb4 modeling
        language!*/
0720.
0721.
0722. import java.util.*;
0723.
0724. /**
0725.  * Subsections within the hospital.
0726.  */
0727. // line 11 "model.ump"
0728. // line 129 "model.ump"
0729. public class Ward
0730. {
0731.
0732.   //------------------------
0733.   // MEMBER VARIABLES
0734.   //------------------------
0735.
0736.   //Ward Attributes
0737.   private String name;
0738.   private int capacity;
0739.
0740.   //Ward Associations
0741.   private List<Employee> employees;
0742.   private List<Patient> patients;
0743.   private Hospital hospital;
0744.   private Inspector inspector;
0745.
0746.   //------------------------
0747.   // CONSTRUCTOR
0748.   //------------------------
0749.
0750.   public Ward(String aName, int aCapacity, Hospital aHospital, Inspector
          aInspector)
0751.   {
0752.     name = aName;
0753.     capacity = aCapacity;
0754.     employees = new ArrayList<Employee>();
0755.     patients = new ArrayList<Patient>();
0756.     boolean didAddHospital = setHospital(aHospital);
0757.     if (!didAddHospital)
0758.     {
0759.       throw new RuntimeException("Unable to create ward due to hospital. See
             http://manual.umple.org?RE002ViolationofAssociationMultiplicity.html");
0760.     }
0761.     boolean didAddInspector = setInspector(aInspector);
0762.     if (!didAddInspector)
0763.     {
0764.       throw new RuntimeException("Unable to create ward due to inspector. See
             http://manual.umple.org?RE002ViolationofAssociationMultiplicity.html");
0765.     }
0766.   }
0767.
0768.   //------------------------
0769.   // INTERFACE
```

```
0770.   //------------------------
0771.
0772.   public boolean setName(String aName)
0773.   {
0774.     boolean wasSet = false;
0775.     name = aName;
0776.     wasSet = true;
0777.     return wasSet;
0778.   }
0779.
0780.   public boolean setCapacity(int aCapacity)
0781.   {
0782.     boolean wasSet = false;
0783.     capacity = aCapacity;
0784.     wasSet = true;
0785.     return wasSet;
0786.   }
0787.
0788.   public String getName()
0789.   {
0790.     return name;
0791.   }
0792.
0793.   public int getCapacity()
0794.   {
0795.     return capacity;
0796.   }
0797.   /* Code from template association_GetMany */
0798.   public Employee getEmployee(int index)
0799.   {
0800.     Employee aEmployee = employees.get(index);
0801.     return aEmployee;
0802.   }
0803.
0804.   public List<Employee> getEmployees()
0805.   {
0806.     List<Employee> newEmployees = Collections.unmodifiableList(employees);
0807.     return newEmployees;
0808.   }
0809.
0810.   public int numberOfEmployees()
0811.   {
0812.     int number = employees.size();
0813.     return number;
0814.   }
0815.
0816.   public boolean hasEmployees()
0817.   {
0818.     boolean has = employees.size() > 0;
0819.     return has;
0820.   }
0821.
0822.   public int indexOfEmployee(Employee aEmployee)
0823.   {
0824.     int index = employees.indexOf(aEmployee);
0825.     return index;
0826.   }
0827.   /* Code from template association_GetMany */
0828.   public Patient getPatient(int index)
0829.   {
0830.     Patient aPatient = patients.get(index);
```

```
0831.     return aPatient;
0832.   }
0833.
0834.   public List<Patient> getPatients()
0835.   {
0836.     List<Patient> newPatients = Collections.unmodifiableList(patients);
0837.     return newPatients;
0838.   }
0839.
0840.   public int numberOfPatients()
0841.   {
0842.     int number = patients.size();
0843.     return number;
0844.   }
0845.
0846.   public boolean hasPatients()
0847.   {
0848.     boolean has = patients.size() > 0;
0849.     return has;
0850.   }
0851.
0852.   public int indexOfPatient(Patient aPatient)
0853.   {
0854.     int index = patients.indexOf(aPatient);
0855.     return index;
0856.   }
0857.   /* Code from template association_GetOne */
0858.   public Hospital getHospital()
0859.   {
0860.     return hospital;
0861.   }
0862.   /* Code from template association_GetOne */
0863.   public Inspector getInspector()
0864.   {
0865.     return inspector;
0866.   }
0867.   /* Code from template association_MinimumNumberOfMethod */
0868.   public static int minimumNumberOfEmployees()
0869.   {
0870.     return 0;
0871.   }
0872.   /* Code from template association_AddManyToManyMethod */
0873.   public boolean addEmployee(Employee aEmployee)
0874.   {
0875.     boolean wasAdded = false;
0876.     if (employees.contains(aEmployee)) { return false; }
0877.     employees.add(aEmployee);
0878.     if (aEmployee.indexOfWard(this) != -1)
0879.     {
0880.       wasAdded = true;
0881.     }
0882.     else
0883.     {
0884.       wasAdded = aEmployee.addWard(this);
0885.       if (!wasAdded)
0886.       {
0887.         employees.remove(aEmployee);
0888.       }
0889.     }
0890.     return wasAdded;
0891.   }
```

```
0892.   /* Code from template association_RemoveMany */
0893.   public boolean removeEmployee(Employee aEmployee)
0894.   {
0895.     boolean wasRemoved = false;
0896.     if (!employees.contains(aEmployee))
0897.     {
0898.       return wasRemoved;
0899.     }
0900.
0901.     int oldIndex = employees.indexOf(aEmployee);
0902.     employees.remove(oldIndex);
0903.     if (aEmployee.indexOfWard(this) == -1)
0904.     {
0905.       wasRemoved = true;
0906.     }
0907.     else
0908.     {
0909.       wasRemoved = aEmployee.removeWard(this);
0910.       if (!wasRemoved)
0911.       {
0912.         employees.add(oldIndex,aEmployee);
0913.       }
0914.     }
0915.     return wasRemoved;
0916.   }
0917.   /* Code from template association_AddIndexControlFunctions */
0918.   public boolean addEmployeeAt(Employee aEmployee, int index)
0919.   {
0920.     boolean wasAdded = false;
0921.     if(addEmployee(aEmployee))
0922.     {
0923.       if(index < 0 ) { index = 0; }
0924.       if(index > numberOfEmployees()) { index = numberOfEmployees() - 1; }
0925.       employees.remove(aEmployee);
0926.       employees.add(index, aEmployee);
0927.       wasAdded = true;
0928.     }
0929.     return wasAdded;
0930.   }
0931.
0932.   public boolean addOrMoveEmployeeAt(Employee aEmployee, int index)
0933.   {
0934.     boolean wasAdded = false;
0935.     if(employees.contains(aEmployee))
0936.     {
0937.       if(index < 0 ) { index = 0; }
0938.       if(index > numberOfEmployees()) { index = numberOfEmployees() - 1; }
0939.       employees.remove(aEmployee);
0940.       employees.add(index, aEmployee);
0941.       wasAdded = true;
0942.     }
0943.     else
0944.     {
0945.       wasAdded = addEmployeeAt(aEmployee, index);
0946.     }
0947.     return wasAdded;
0948.   }
0949.   /* Code from template association_MinimumNumberOfMethod */
0950.   public static int minimumNumberOfPatients()
0951.   {
0952.     return 0;
```

```
0953.   }
0954.   /* Code from template association_AddManyToOne */
0955.   public Patient addPatient(String aName)
0956.   {
0957.     return new Patient(aName, this);
0958.   }
0959.
0960.   public boolean addPatient(Patient aPatient)
0961.   {
0962.     boolean wasAdded = false;
0963.     if (patients.contains(aPatient)) { return false; }
0964.     Ward existingWard = aPatient.getWard();
0965.     boolean isNewWard = existingWard != null && !this.equals(existingWard);
0966.     if (isNewWard)
0967.     {
0968.       aPatient.setWard(this);
0969.     }
0970.     else
0971.     {
0972.       patients.add(aPatient);
0973.     }
0974.     wasAdded = true;
0975.     return wasAdded;
0976.   }
0977.
0978.   public boolean removePatient(Patient aPatient)
0979.   {
0980.     boolean wasRemoved = false;
0981.     //Unable to remove aPatient, as it must always have a ward
0982.     if (!this.equals(aPatient.getWard()))
0983.     {
0984.       patients.remove(aPatient);
0985.       wasRemoved = true;
0986.     }
0987.     return wasRemoved;
0988.   }
0989.   /* Code from template association_AddIndexControlFunctions */
0990.   public boolean addPatientAt(Patient aPatient, int index)
0991.   {
0992.     boolean wasAdded = false;
0993.     if(addPatient(aPatient))
0994.     {
0995.       if(index < 0 ) { index = 0; }
0996.       if(index > numberOfPatients()) { index = numberOfPatients() - 1; }
0997.       patients.remove(aPatient);
0998.       patients.add(index, aPatient);
0999.       wasAdded = true;
1000.    }
1001.    return wasAdded;
1002.  }
1003.
1004.  public boolean addOrMovePatientAt(Patient aPatient, int index)
1005.  {
1006.    boolean wasAdded = false;
1007.    if(patients.contains(aPatient))
1008.    {
1009.      if(index < 0 ) { index = 0; }
1010.      if(index > numberOfPatients()) { index = numberOfPatients() - 1; }
1011.      patients.remove(aPatient);
1012.      patients.add(index, aPatient);
1013.      wasAdded = true;
```

```
1014.     }
1015.     else
1016.     {
1017.       wasAdded = addPatientAt(aPatient, index);
1018.     }
1019.     return wasAdded;
1020.   }
1021.   /* Code from template association_SetOneToMandatoryMany */
1022.   public boolean setHospital(Hospital aHospital)
1023.   {
1024.     boolean wasSet = false;
1025.     //Must provide hospital to ward
1026.     if (aHospital == null)
1027.     {
1028.       return wasSet;
1029.     }
1030.
1031.     if (hospital != null && hospital.numberOfWards() <=
            Hospital.minimumNumberOfWards())
1032.     {
1033.       return wasSet;
1034.     }
1035.
1036.     Hospital existingHospital = hospital;
1037.     hospital = aHospital;
1038.     if (existingHospital != null && !existingHospital.equals(aHospital))
1039.     {
1040.       boolean didRemove = existingHospital.removeWard(this);
1041.       if (!didRemove)
1042.       {
1043.         hospital = existingHospital;
1044.         return wasSet;
1045.       }
1046.     }
1047.     hospital.addWard(this);
1048.     wasSet = true;
1049.     return wasSet;
1050.   }
1051.   /* Code from template association_SetOneToMany */
1052.   public boolean setInspector(Inspector aInspector)
1053.   {
1054.     boolean wasSet = false;
1055.     if (aInspector == null)
1056.     {
1057.       return wasSet;
1058.     }
1059.
1060.     Inspector existingInspector = inspector;
1061.     inspector = aInspector;
1062.     if (existingInspector != null && !existingInspector.equals(aInspector))
1063.     {
1064.       existingInspector.removeWard(this);
1065.     }
1066.     inspector.addWard(this);
1067.     wasSet = true;
1068.     return wasSet;
1069.   }
1070.
1071.   public void delete()
1072.   {
1073.     ArrayList<Employee> copyOfEmployees = new ArrayList<Employee>(employees);
```

```
1074.     employees.clear();
1075.     for(Employee aEmployee : copyOfEmployees)
1076.     {
1077.       if (aEmployee.numberOfWards() <= Employee.minimumNumberOfWards())
1078.       {
1079.         aEmployee.delete();
1080.       }
1081.       else
1082.       {
1083.         aEmployee.removeWard(this);
1084.       }
1085.     }
1086.     for(int i=patients.size(); i > 0; i--)
1087.     {
1088.       Patient aPatient = patients.get(i - 1);
1089.       aPatient.delete();
1090.     }
1091.     Hospital placeholderHospital = hospital;
1092.     this.hospital = null;
1093.     if(placeholderHospital != null)
1094.     {
1095.       placeholderHospital.removeWard(this);
1096.     }
1097.     Inspector placeholderInspector = inspector;
1098.     this.inspector = null;
1099.     if(placeholderInspector != null)
1100.     {
1101.       placeholderInspector.removeWard(this);
1102.     }
1103.   }
1104.
1105.
1106.   public String toString()
1107.   {
1108.     return super.toString() + "["+
1109.            "name" + ":" + getName()+ "," +
1110.            "capacity" + ":" + getCapacity()+ "]" +
                  System.getProperties().getProperty("line.separator") +
1111.            "  " + "hospital = "+(getHospital()!=null?
                  Integer.toHexString(System.identityHashCode(getHospital())):"null")
                  + System.getProperties().getProperty("line.separator") +
1112.            "  " + "inspector = "+(getInspector()!=null?
                  Integer.toHexString(System.identityHashCode(getInspector())):"null");
1113.   }
1114. }
1115.
1116.
1117.
1118. //%% NEW FILE Janitor BEGINS HERE %%
1119.
1120. /*PLEASE DO NOT EDIT THIS CODE*/
1121. /*This code was generated using the UMPLE 1.29.1.4787.f023c4bb4 modeling
       language!*/
1122.
1123.
1124. import java.util.*;
1125.
1126. /**
1127.  * Employee who maintains the cleanliness of the hospital.
1128.  */
1129. // line 67 "model.ump"
```

```
1130. // line 100 "model.ump"
1131. public class Janitor extends Employee
1132. {
1133. 
1134.   //------------------------
1135.   // MEMBER VARIABLES
1136.   //------------------------
1137. 
1138.   //------------------------
1139.   // CONSTRUCTOR
1140.   //------------------------
1141. 
1142.   public Janitor(String aName, int aId, int aSalary, Hospital aHospital, Ward...
          allWards)
1143.   {
1144.     super(aName, aId, aSalary, aHospital, allWards);
1145.   }
1146. 
1147.   //------------------------
1148.   // INTERFACE
1149.   //------------------------
1150. 
1151.   public void delete()
1152.   {
1153.     super.delete();
1154.   }
1155. 
1156. }
1157. 
1158. 
1159. 
1160. //%% NEW FILE Inspector BEGINS HERE %%
1161. 
1162. /*PLEASE DO NOT EDIT THIS CODE*/
1163. /*This code was generated using the UMPLE 1.29.1.4787.f023c4bb4 modeling
       language!*/
1164. 
1165. 
1166. import java.util.*;
1167. 
1168. // line 49 "model.ump"
1169. // line 136 "model.ump"
1170. public class Inspector extends Employee
1171. {
1172. 
1173.   //------------------------
1174.   // MEMBER VARIABLES
1175.   //------------------------
1176. 
1177.   //Inspector Associations
1178.   private List<Ward> wards;
1179. 
1180.   //------------------------
1181.   // CONSTRUCTOR
1182.   //------------------------
1183. 
1184.   public Inspector(String aName, int aId, int aSalary, Hospital aHospital,
          Ward... allWards)
1185.   {
1186.     super(aName, aId, aSalary, aHospital, allWards);
1187.     wards = new ArrayList<Ward>();
```

```
1188.   }
1189.
1190.   //------------------------
1191.   // INTERFACE
1192.   //------------------------
1193.   /* Code from template association_GetMany */
1194.   public Ward getWard(int index)
1195.   {
1196.     Ward aWard = wards.get(index);
1197.     return aWard;
1198.   }
1199.
1200.   public List<Ward> getWards()
1201.   {
1202.     List<Ward> newWards = Collections.unmodifiableList(wards);
1203.     return newWards;
1204.   }
1205.
1206.   public int numberOfWards()
1207.   {
1208.     int number = wards.size();
1209.     return number;
1210.   }
1211.
1212.   public boolean hasWards()
1213.   {
1214.     boolean has = wards.size() > 0;
1215.     return has;
1216.   }
1217.
1218.   public int indexOfWard(Ward aWard)
1219.   {
1220.     int index = wards.indexOf(aWard);
1221.     return index;
1222.   }
1223.   /* Code from template association_MinimumNumberOfMethod */
1224.   public static int minimumNumberOfWards()
1225.   {
1226.     return 0;
1227.   }
1228.   /* Code from template association_AddManyToOne */
1229.   public Ward addWard(String aName, int aCapacity, Hospital aHospital)
1230.   {
1231.     return new Ward(aName, aCapacity, aHospital, this);
1232.   }
1233.
1234.   public boolean addWard(Ward aWard)
1235.   {
1236.     boolean wasAdded = false;
1237.     if (wards.contains(aWard)) { return false; }
1238.     Inspector existingInspector = aWard.getInspector();
1239.     boolean isNewInspector = existingInspector != null &&
          !this.equals(existingInspector);
1240.     if (isNewInspector)
1241.     {
1242.       aWard.setInspector(this);
1243.     }
1244.     else
1245.     {
1246.       wards.add(aWard);
1247.     }
```

```
1248.     wasAdded = true;
1249.     return wasAdded;
1250.   }
1251.
1252.   public boolean removeWard(Ward aWard)
1253.   {
1254.     boolean wasRemoved = false;
1255.     //Unable to remove aWard, as it must always have a inspector
1256.     if (!this.equals(aWard.getInspector()))
1257.     {
1258.       wards.remove(aWard);
1259.       wasRemoved = true;
1260.     }
1261.     return wasRemoved;
1262.   }
1263.   /* Code from template association_AddIndexControlFunctions */
1264.   public boolean addWardAt(Ward aWard, int index)
1265.   {
1266.     boolean wasAdded = false;
1267.     if(addWard(aWard))
1268.     {
1269.       if(index < 0 ) { index = 0; }
1270.       if(index > numberOfWards()) { index = numberOfWards() - 1; }
1271.       wards.remove(aWard);
1272.       wards.add(index, aWard);
1273.       wasAdded = true;
1274.     }
1275.     return wasAdded;
1276.   }
1277.
1278.   public boolean addOrMoveWardAt(Ward aWard, int index)
1279.   {
1280.     boolean wasAdded = false;
1281.     if(wards.contains(aWard))
1282.     {
1283.       if(index < 0 ) { index = 0; }
1284.       if(index > numberOfWards()) { index = numberOfWards() - 1; }
1285.       wards.remove(aWard);
1286.       wards.add(index, aWard);
1287.       wasAdded = true;
1288.     }
1289.     else
1290.     {
1291.       wasAdded = addWardAt(aWard, index);
1292.     }
1293.     return wasAdded;
1294.   }
1295.
1296.   public void delete()
1297.   {
1298.     for(int i=wards.size(); i > 0; i--)
1299.     {
1300.       Ward aWard = wards.get(i - 1);
1301.       aWard.delete();
1302.     }
1303.     super.delete();
1304.   }
1305.
1306. }
1307.
1308.
```

```
1309.
1310. //%% NEW FILE Patient BEGINS HERE %%
1311.
1312. /*PLEASE DO NOT EDIT THIS CODE*/
1313. /*This code was generated using the UMPLE 1.29.1.4787.f023c4bb4 modeling
        language!*/
1314.
1315.
1316. import java.util.*;
1317.
1318. /**
1319.  * Patient who is at the hospital to get better.
1320.  */
1321. // line 73 "model.ump"
1322. // line 105 "model.ump"
1323. public class Patient
1324. {
1325.
1326.   //------------------------
1327.   // MEMBER VARIABLES
1328.   //------------------------
1329.
1330.   //Patient Attributes
1331.   private String name;
1332.
1333.   //Patient Associations
1334.   private List<Doctor> doctors;
1335.   private Ward ward;
1336.   private List<Nurse> nurses;
1337.
1338.   //------------------------
1339.   // CONSTRUCTOR
1340.   //------------------------
1341.
1342.   public Patient(String aName, Ward aWard)
1343.   {
1344.     name = aName;
1345.     doctors = new ArrayList<Doctor>();
1346.     boolean didAddWard = setWard(aWard);
1347.     if (!didAddWard)
1348.     {
1349.       throw new RuntimeException("Unable to create patient due to ward. See
            http://manual.umple.org?RE002ViolationofAssociationMultiplicity.html");
1350.     }
1351.     nurses = new ArrayList<Nurse>();
1352.   }
1353.
1354.   //------------------------
1355.   // INTERFACE
1356.   //------------------------
1357.
1358.   public boolean setName(String aName)
1359.   {
1360.     boolean wasSet = false;
1361.     name = aName;
1362.     wasSet = true;
1363.     return wasSet;
1364.   }
1365.
1366.   public String getName()
1367.   {
```

```
1368.     return name;
1369.   }
1370.   /* Code from template association_GetMany */
1371.   public Doctor getDoctor(int index)
1372.   {
1373.     Doctor aDoctor = doctors.get(index);
1374.     return aDoctor;
1375.   }
1376.
1377.   public List<Doctor> getDoctors()
1378.   {
1379.     List<Doctor> newDoctors = Collections.unmodifiableList(doctors);
1380.     return newDoctors;
1381.   }
1382.
1383.   public int numberOfDoctors()
1384.   {
1385.     int number = doctors.size();
1386.     return number;
1387.   }
1388.
1389.   public boolean hasDoctors()
1390.   {
1391.     boolean has = doctors.size() > 0;
1392.     return has;
1393.   }
1394.
1395.   public int indexOfDoctor(Doctor aDoctor)
1396.   {
1397.     int index = doctors.indexOf(aDoctor);
1398.     return index;
1399.   }
1400.   /* Code from template association_GetOne */
1401.   public Ward getWard()
1402.   {
1403.     return ward;
1404.   }
1405.   /* Code from template association_GetMany */
1406.   public Nurse getNurse(int index)
1407.   {
1408.     Nurse aNurse = nurses.get(index);
1409.     return aNurse;
1410.   }
1411.
1412.   public List<Nurse> getNurses()
1413.   {
1414.     List<Nurse> newNurses = Collections.unmodifiableList(nurses);
1415.     return newNurses;
1416.   }
1417.
1418.   public int numberOfNurses()
1419.   {
1420.     int number = nurses.size();
1421.     return number;
1422.   }
1423.
1424.   public boolean hasNurses()
1425.   {
1426.     boolean has = nurses.size() > 0;
1427.     return has;
1428.   }
```

```
1429.
1430.   public int indexOfNurse(Nurse aNurse)
1431.   {
1432.     int index = nurses.indexOf(aNurse);
1433.     return index;
1434.   }
1435.   /* Code from template association_MinimumNumberOfMethod */
1436.   public static int minimumNumberOfDoctors()
1437.   {
1438.     return 0;
1439.   }
1440.   /* Code from template association_AddManyToManyMethod */
1441.   public boolean addDoctor(Doctor aDoctor)
1442.   {
1443.     boolean wasAdded = false;
1444.     if (doctors.contains(aDoctor)) { return false; }
1445.     doctors.add(aDoctor);
1446.     if (aDoctor.indexOfPatient(this) != -1)
1447.     {
1448.       wasAdded = true;
1449.     }
1450.     else
1451.     {
1452.       wasAdded = aDoctor.addPatient(this);
1453.       if (!wasAdded)
1454.       {
1455.         doctors.remove(aDoctor);
1456.       }
1457.     }
1458.     return wasAdded;
1459.   }
1460.   /* Code from template association_RemoveMany */
1461.   public boolean removeDoctor(Doctor aDoctor)
1462.   {
1463.     boolean wasRemoved = false;
1464.     if (!doctors.contains(aDoctor))
1465.     {
1466.       return wasRemoved;
1467.     }
1468.
1469.     int oldIndex = doctors.indexOf(aDoctor);
1470.     doctors.remove(oldIndex);
1471.     if (aDoctor.indexOfPatient(this) == -1)
1472.     {
1473.       wasRemoved = true;
1474.     }
1475.     else
1476.     {
1477.       wasRemoved = aDoctor.removePatient(this);
1478.       if (!wasRemoved)
1479.       {
1480.         doctors.add(oldIndex,aDoctor);
1481.       }
1482.     }
1483.     return wasRemoved;
1484.   }
1485.   /* Code from template association_AddIndexControlFunctions */
1486.   public boolean addDoctorAt(Doctor aDoctor, int index)
1487.   {
1488.     boolean wasAdded = false;
1489.     if(addDoctor(aDoctor))
```

```
1490.     {
1491.       if(index < 0 ) { index = 0; }
1492.       if(index > numberOfDoctors()) { index = numberOfDoctors() - 1; }
1493.       doctors.remove(aDoctor);
1494.       doctors.add(index, aDoctor);
1495.       wasAdded = true;
1496.     }
1497.     return wasAdded;
1498.   }
1499.
1500.   public boolean addOrMoveDoctorAt(Doctor aDoctor, int index)
1501.   {
1502.     boolean wasAdded = false;
1503.     if(doctors.contains(aDoctor))
1504.     {
1505.       if(index < 0 ) { index = 0; }
1506.       if(index > numberOfDoctors()) { index = numberOfDoctors() - 1; }
1507.       doctors.remove(aDoctor);
1508.       doctors.add(index, aDoctor);
1509.       wasAdded = true;
1510.     }
1511.     else
1512.     {
1513.       wasAdded = addDoctorAt(aDoctor, index);
1514.     }
1515.     return wasAdded;
1516.   }
1517.   /* Code from template association_SetOneToMany */
1518.   public boolean setWard(Ward aWard)
1519.   {
1520.     boolean wasSet = false;
1521.     if (aWard == null)
1522.     {
1523.       return wasSet;
1524.     }
1525.
1526.     Ward existingWard = ward;
1527.     ward = aWard;
1528.     if (existingWard != null && !existingWard.equals(aWard))
1529.     {
1530.       existingWard.removePatient(this);
1531.     }
1532.     ward.addPatient(this);
1533.     wasSet = true;
1534.     return wasSet;
1535.   }
1536.   /* Code from template association_MinimumNumberOfMethod */
1537.   public static int minimumNumberOfNurses()
1538.   {
1539.     return 0;
1540.   }
1541.   /* Code from template association_AddManyToManyMethod */
1542.   public boolean addNurse(Nurse aNurse)
1543.   {
1544.     boolean wasAdded = false;
1545.     if (nurses.contains(aNurse)) { return false; }
1546.     nurses.add(aNurse);
1547.     if (aNurse.indexOfPatient(this) != -1)
1548.     {
1549.       wasAdded = true;
1550.     }
```

```
    else
    {
      wasAdded = aNurse.addPatient(this);
      if (!wasAdded)
      {
        nurses.remove(aNurse);
      }
    }
    return wasAdded;
  }
  /* Code from template association_RemoveMany */
  public boolean removeNurse(Nurse aNurse)
  {
    boolean wasRemoved = false;
    if (!nurses.contains(aNurse))
    {
      return wasRemoved;
    }

    int oldIndex = nurses.indexOf(aNurse);
    nurses.remove(oldIndex);
    if (aNurse.indexOfPatient(this) == -1)
    {
      wasRemoved = true;
    }
    else
    {
      wasRemoved = aNurse.removePatient(this);
      if (!wasRemoved)
      {
        nurses.add(oldIndex,aNurse);
      }
    }
    return wasRemoved;
  }
  /* Code from template association_AddIndexControlFunctions */
  public boolean addNurseAt(Nurse aNurse, int index)
  {
    boolean wasAdded = false;
    if(addNurse(aNurse))
    {
      if(index < 0 ) { index = 0; }
      if(index > numberOfNurses()) { index = numberOfNurses() - 1; }
      nurses.remove(aNurse);
      nurses.add(index, aNurse);
      wasAdded = true;
    }
    return wasAdded;
  }

  public boolean addOrMoveNurseAt(Nurse aNurse, int index)
  {
    boolean wasAdded = false;
    if(nurses.contains(aNurse))
    {
      if(index < 0 ) { index = 0; }
      if(index > numberOfNurses()) { index = numberOfNurses() - 1; }
      nurses.remove(aNurse);
      nurses.add(index, aNurse);
      wasAdded = true;
    }
```

```
1612.     else
1613.     {
1614.       wasAdded = addNurseAt(aNurse, index);
1615.     }
1616.     return wasAdded;
1617.   }
1618.
1619.   public void delete()
1620.   {
1621.     ArrayList<Doctor> copyOfDoctors = new ArrayList<Doctor>(doctors);
1622.     doctors.clear();
1623.     for(Doctor aDoctor : copyOfDoctors)
1624.     {
1625.       aDoctor.removePatient(this);
1626.     }
1627.     Ward placeholderWard = ward;
1628.     this.ward = null;
1629.     if(placeholderWard != null)
1630.     {
1631.       placeholderWard.removePatient(this);
1632.     }
1633.     ArrayList<Nurse> copyOfNurses = new ArrayList<Nurse>(nurses);
1634.     nurses.clear();
1635.     for(Nurse aNurse : copyOfNurses)
1636.     {
1637.       aNurse.removePatient(this);
1638.     }
1639.   }
1640.
1641.
1642.   public String toString()
1643.   {
1644.     return super.toString() + "["+
1645.            "name" + ":" + getName()+ "]" +
                   System.getProperties().getProperty("line.separator") +
1646.            "  " + "ward = "+(getWard()!=null?
                   Integer.toHexString(System.identityHashCode(getWard())):"null");
1647.   }
1648. }
1649.
1650.
1651.
1652. //%% NEW FILE Hospital BEGINS HERE %%
1653.
1654. /*PLEASE DO NOT EDIT THIS CODE*/
1655. /*This code was generated using the UMPLE 1.29.1.4787.f023c4bb4 modeling
       language!*/
1656.
1657.
1658. import java.util.*;
1659.
1660. /**
1661.  * Hospital system - sample UML class diagram in Umple
1662.  */
1663. // line 4 "model.ump"
1664. // line 83 "model.ump"
1665. public class Hospital
1666. {
1667.
1668.   //------------------------
1669.   // MEMBER VARIABLES
```

```
1670.   //------------------------
1671.
1672.   //Hospital Associations
1673.   private List<Employee> employees;
1674.   private List<Ward> wards;
1675.
1676.   //------------------------
1677.   // CONSTRUCTOR
1678.   //------------------------
1679.
1680.   public Hospital()
1681.   {
1682.     employees = new ArrayList<Employee>();
1683.     wards = new ArrayList<Ward>();
1684.   }
1685.
1686.   //------------------------
1687.   // INTERFACE
1688.   //------------------------
1689.   /* Code from template association_GetMany */
1690.   public Employee getEmployee(int index)
1691.   {
1692.     Employee aEmployee = employees.get(index);
1693.     return aEmployee;
1694.   }
1695.
1696.   public List<Employee> getEmployees()
1697.   {
1698.     List<Employee> newEmployees = Collections.unmodifiableList(employees);
1699.     return newEmployees;
1700.   }
1701.
1702.   public int numberOfEmployees()
1703.   {
1704.     int number = employees.size();
1705.     return number;
1706.   }
1707.
1708.   public boolean hasEmployees()
1709.   {
1710.     boolean has = employees.size() > 0;
1711.     return has;
1712.   }
1713.
1714.   public int indexOfEmployee(Employee aEmployee)
1715.   {
1716.     int index = employees.indexOf(aEmployee);
1717.     return index;
1718.   }
1719.   /* Code from template association_GetMany */
1720.   public Ward getWard(int index)
1721.   {
1722.     Ward aWard = wards.get(index);
1723.     return aWard;
1724.   }
1725.
1726.   public List<Ward> getWards()
1727.   {
1728.     List<Ward> newWards = Collections.unmodifiableList(wards);
1729.     return newWards;
1730.   }
```

```
1731.
1732.   public int numberOfWards()
1733.   {
1734.     int number = wards.size();
1735.     return number;
1736.   }
1737.
1738.   public boolean hasWards()
1739.   {
1740.     boolean has = wards.size() > 0;
1741.     return has;
1742.   }
1743.
1744.   public int indexOfWard(Ward aWard)
1745.   {
1746.     int index = wards.indexOf(aWard);
1747.     return index;
1748.   }
1749.   /* Code from template association_MinimumNumberOfMethod */
1750.   public static int minimumNumberOfEmployees()
1751.   {
1752.     return 0;
1753.   }
1754.   /* Code from template association_AddManyToOne */
1755.   public Employee addEmployee(String aName, int aId, int aSalary, Ward...
        allWards)
1756.   {
1757.     return new Employee(aName, aId, aSalary, this, allWards);
1758.   }
1759.
1760.   public boolean addEmployee(Employee aEmployee)
1761.   {
1762.     boolean wasAdded = false;
1763.     if (employees.contains(aEmployee)) { return false; }
1764.     Hospital existingHospital = aEmployee.getHospital();
1765.     boolean isNewHospital = existingHospital != null &&
          !this.equals(existingHospital);
1766.     if (isNewHospital)
1767.     {
1768.       aEmployee.setHospital(this);
1769.     }
1770.     else
1771.     {
1772.       employees.add(aEmployee);
1773.     }
1774.     wasAdded = true;
1775.     return wasAdded;
1776.   }
1777.
1778.   public boolean removeEmployee(Employee aEmployee)
1779.   {
1780.     boolean wasRemoved = false;
1781.     //Unable to remove aEmployee, as it must always have a hospital
1782.     if (!this.equals(aEmployee.getHospital()))
1783.     {
1784.       employees.remove(aEmployee);
1785.       wasRemoved = true;
1786.     }
1787.     return wasRemoved;
1788.   }
1789.   /* Code from template association_AddIndexControlFunctions */
```

```
1790.   public boolean addEmployeeAt(Employee aEmployee, int index)
1791.   {
1792.     boolean wasAdded = false;
1793.     if(addEmployee(aEmployee))
1794.     {
1795.       if(index < 0 ) { index = 0; }
1796.       if(index > numberOfEmployees()) { index = numberOfEmployees() - 1; }
1797.       employees.remove(aEmployee);
1798.       employees.add(index, aEmployee);
1799.       wasAdded = true;
1800.     }
1801.     return wasAdded;
1802.   }
1803.
1804.   public boolean addOrMoveEmployeeAt(Employee aEmployee, int index)
1805.   {
1806.     boolean wasAdded = false;
1807.     if(employees.contains(aEmployee))
1808.     {
1809.       if(index < 0 ) { index = 0; }
1810.       if(index > numberOfEmployees()) { index = numberOfEmployees() - 1; }
1811.       employees.remove(aEmployee);
1812.       employees.add(index, aEmployee);
1813.       wasAdded = true;
1814.     }
1815.     else
1816.     {
1817.       wasAdded = addEmployeeAt(aEmployee, index);
1818.     }
1819.     return wasAdded;
1820.   }
1821.   /* Code from template association_IsNumberOfValidMethod */
1822.   public boolean isNumberOfWardsValid()
1823.   {
1824.     boolean isValid = numberOfWards() >= minimumNumberOfWards();
1825.     return isValid;
1826.   }
1827.   /* Code from template association_MinimumNumberOfMethod */
1828.   public static int minimumNumberOfWards()
1829.   {
1830.     return 1;
1831.   }
1832.   /* Code from template association_AddMandatoryManyToOne */
1833.   public Ward addWard(String aName, int aCapacity, Inspector aInspector)
1834.   {
1835.     Ward aNewWard = new Ward(aName, aCapacity, this, aInspector);
1836.     return aNewWard;
1837.   }
1838.
1839.   public boolean addWard(Ward aWard)
1840.   {
1841.     boolean wasAdded = false;
1842.     if (wards.contains(aWard)) { return false; }
1843.     Hospital existingHospital = aWard.getHospital();
1844.     boolean isNewHospital = existingHospital != null &&
          !this.equals(existingHospital);
1845.
1846.     if (isNewHospital && existingHospital.numberOfWards() <=
          minimumNumberOfWards())
1847.     {
1848.       return wasAdded;
```

```
1849.     }
1850.     if (isNewHospital)
1851.     {
1852.       aWard.setHospital(this);
1853.     }
1854.     else
1855.     {
1856.       wards.add(aWard);
1857.     }
1858.     wasAdded = true;
1859.     return wasAdded;
1860.   }
1861.
1862.   public boolean removeWard(Ward aWard)
1863.   {
1864.     boolean wasRemoved = false;
1865.     //Unable to remove aWard, as it must always have a hospital
1866.     if (this.equals(aWard.getHospital()))
1867.     {
1868.       return wasRemoved;
1869.     }
1870.
1871.     //hospital already at minimum (1)
1872.     if (numberOfWards() <= minimumNumberOfWards())
1873.     {
1874.       return wasRemoved;
1875.     }
1876.
1877.     wards.remove(aWard);
1878.     wasRemoved = true;
1879.     return wasRemoved;
1880.   }
1881.   /* Code from template association_AddIndexControlFunctions */
1882.   public boolean addWardAt(Ward aWard, int index)
1883.   {
1884.     boolean wasAdded = false;
1885.     if(addWard(aWard))
1886.     {
1887.       if(index < 0 ) { index = 0; }
1888.       if(index > numberOfWards()) { index = numberOfWards() - 1; }
1889.       wards.remove(aWard);
1890.       wards.add(index, aWard);
1891.       wasAdded = true;
1892.     }
1893.     return wasAdded;
1894.   }
1895.
1896.   public boolean addOrMoveWardAt(Ward aWard, int index)
1897.   {
1898.     boolean wasAdded = false;
1899.     if(wards.contains(aWard))
1900.     {
1901.       if(index < 0 ) { index = 0; }
1902.       if(index > numberOfWards()) { index = numberOfWards() - 1; }
1903.       wards.remove(aWard);
1904.       wards.add(index, aWard);
1905.       wasAdded = true;
1906.     }
1907.     else
1908.     {
1909.       wasAdded = addWardAt(aWard, index);
```

```
    }
    return wasAdded;
  }

  public void delete()
  {
    for(int i=employees.size(); i > 0; i--)
    {
      Employee aEmployee = employees.get(i - 1);
      aEmployee.delete();
    }
    for(int i=wards.size(); i > 0; i--)
    {
      Ward aWard = wards.get(i - 1);
      aWard.delete();
    }
  }

}



//%% NEW FILE Doctor BEGINS HERE %%

/*PLEASE DO NOT EDIT THIS CODE*/
/*This code was generated using the UMPLE 1.29.1.4787.f023c4bb4 modeling
   language!*/


import java.util.*;

/**
 * Specialized employee who looks after patients.
 */
// line 44 "model.ump"
// line 90 "model.ump"
public class Doctor extends Employee
{

  //------------------------
  // MEMBER VARIABLES
  //------------------------

  //Doctor Associations
  private List<Patient> patients;

  //------------------------
  // CONSTRUCTOR
  //------------------------

  public Doctor(String aName, int aId, int aSalary, Hospital aHospital, Ward...
     allWards)
  {
    super(aName, aId, aSalary, aHospital, allWards);
    patients = new ArrayList<Patient>();
  }

  //------------------------
  // INTERFACE
  //------------------------
  /* Code from template association_GetMany */
```

```
1969.   public Patient getPatient(int index)
1970.   {
1971.     Patient aPatient = patients.get(index);
1972.     return aPatient;
1973.   }
1974.
1975.   public List<Patient> getPatients()
1976.   {
1977.     List<Patient> newPatients = Collections.unmodifiableList(patients);
1978.     return newPatients;
1979.   }
1980.
1981.   public int numberOfPatients()
1982.   {
1983.     int number = patients.size();
1984.     return number;
1985.   }
1986.
1987.   public boolean hasPatients()
1988.   {
1989.     boolean has = patients.size() > 0;
1990.     return has;
1991.   }
1992.
1993.   public int indexOfPatient(Patient aPatient)
1994.   {
1995.     int index = patients.indexOf(aPatient);
1996.     return index;
1997.   }
1998.   /* Code from template association_MinimumNumberOfMethod */
1999.   public static int minimumNumberOfPatients()
2000.   {
2001.     return 0;
2002.   }
2003.   /* Code from template association_AddManyToManyMethod */
2004.   public boolean addPatient(Patient aPatient)
2005.   {
2006.     boolean wasAdded = false;
2007.     if (patients.contains(aPatient)) { return false; }
2008.     patients.add(aPatient);
2009.     if (aPatient.indexOfDoctor(this) != -1)
2010.     {
2011.       wasAdded = true;
2012.     }
2013.     else
2014.     {
2015.       wasAdded = aPatient.addDoctor(this);
2016.       if (!wasAdded)
2017.       {
2018.         patients.remove(aPatient);
2019.       }
2020.     }
2021.     return wasAdded;
2022.   }
2023.   /* Code from template association_RemoveMany */
2024.   public boolean removePatient(Patient aPatient)
2025.   {
2026.     boolean wasRemoved = false;
2027.     if (!patients.contains(aPatient))
2028.     {
2029.       return wasRemoved;
```

```
2030.     }
2031.
2032.     int oldIndex = patients.indexOf(aPatient);
2033.     patients.remove(oldIndex);
2034.     if (aPatient.indexOfDoctor(this) == -1)
2035.     {
2036.       wasRemoved = true;
2037.     }
2038.     else
2039.     {
2040.       wasRemoved = aPatient.removeDoctor(this);
2041.       if (!wasRemoved)
2042.       {
2043.         patients.add(oldIndex,aPatient);
2044.       }
2045.     }
2046.     return wasRemoved;
2047.   }
2048.   /* Code from template association_AddIndexControlFunctions */
2049.   public boolean addPatientAt(Patient aPatient, int index)
2050.   {
2051.     boolean wasAdded = false;
2052.     if(addPatient(aPatient))
2053.     {
2054.       if(index < 0 ) { index = 0; }
2055.       if(index > numberOfPatients()) { index = numberOfPatients() - 1; }
2056.       patients.remove(aPatient);
2057.       patients.add(index, aPatient);
2058.       wasAdded = true;
2059.     }
2060.     return wasAdded;
2061.   }
2062.
2063.   public boolean addOrMovePatientAt(Patient aPatient, int index)
2064.   {
2065.     boolean wasAdded = false;
2066.     if(patients.contains(aPatient))
2067.     {
2068.       if(index < 0 ) { index = 0; }
2069.       if(index > numberOfPatients()) { index = numberOfPatients() - 1; }
2070.       patients.remove(aPatient);
2071.       patients.add(index, aPatient);
2072.       wasAdded = true;
2073.     }
2074.     else
2075.     {
2076.       wasAdded = addPatientAt(aPatient, index);
2077.     }
2078.     return wasAdded;
2079.   }
2080.
2081.   public void delete()
2082.   {
2083.     ArrayList<Patient> copyOfPatients = new ArrayList<Patient>(patients);
2084.     patients.clear();
2085.     for(Patient aPatient : copyOfPatients)
2086.     {
2087.       aPatient.removeDoctor(this);
2088.     }
2089.     super.delete();
2090.   }
```

```
2091.
2092. }
2093.
2094.
2095.
2096. //%% NEW FILE Privilege BEGINS HERE %%
2097.
2098. /*PLEASE DO NOT EDIT THIS CODE*/
2099. /*This code was generated using the UMPLE 1.29.1.4787.f023c4bb4 modeling
        language!*/
2100.
2101.
2102.
2103. /**
2104.  * Various privileges and roles that the employees have.
2105.  */
2106. // line 38 "model.ump"
2107. // line 117 "model.ump"
2108. public class Privilege
2109. {
2110.
2111.   //------------------------
2112.   // MEMBER VARIABLES
2113.   //------------------------
2114.
2115.   //Privilege Attributes
2116.   private String privilege;
2117.
2118.   //Privilege Associations
2119.   private Employee employee;
2120.
2121.   //------------------------
2122.   // CONSTRUCTOR
2123.   //------------------------
2124.
2125.   public Privilege(String aPrivilege, Employee aEmployee)
2126.   {
2127.     privilege = aPrivilege;
2128.     boolean didAddEmployee = setEmployee(aEmployee);
2129.     if (!didAddEmployee)
2130.     {
2131.       throw new RuntimeException("Unable to create privilege due to employee. See
            http://manual.umple.org?RE002ViolationofAssociationMultiplicity.html");
2132.     }
2133.   }
2134.
2135.   //------------------------
2136.   // INTERFACE
2137.   //------------------------
2138.
2139.   public boolean setPrivilege(String aPrivilege)
2140.   {
2141.     boolean wasSet = false;
2142.     privilege = aPrivilege;
2143.     wasSet = true;
2144.     return wasSet;
2145.   }
2146.
2147.   public String getPrivilege()
2148.   {
2149.     return privilege;
```

```
2150.   }
2151.   /* Code from template association_GetOne */
2152.   public Employee getEmployee()
2153.   {
2154.     return employee;
2155.   }
2156.   /* Code from template association_SetOneToMandatoryMany */
2157.   public boolean setEmployee(Employee aEmployee)
2158.   {
2159.     boolean wasSet = false;
2160.     //Must provide employee to privilege
2161.     if (aEmployee == null)
2162.     {
2163.       return wasSet;
2164.     }
2165.
2166.     if (employee != null && employee.numberOfPrivileges() <=
           Employee.minimumNumberOfPrivileges())
2167.     {
2168.       return wasSet;
2169.     }
2170.
2171.     Employee existingEmployee = employee;
2172.     employee = aEmployee;
2173.     if (existingEmployee != null && !existingEmployee.equals(aEmployee))
2174.     {
2175.       boolean didRemove = existingEmployee.removePrivilege(this);
2176.       if (!didRemove)
2177.       {
2178.         employee = existingEmployee;
2179.         return wasSet;
2180.       }
2181.     }
2182.     employee.addPrivilege(this);
2183.     wasSet = true;
2184.     return wasSet;
2185.   }
2186.
2187.   public void delete()
2188.   {
2189.     Employee placeholderEmployee = employee;
2190.     this.employee = null;
2191.     if(placeholderEmployee != null)
2192.     {
2193.       placeholderEmployee.removePrivilege(this);
2194.     }
2195.   }
2196.
2197.
2198.   public String toString()
2199.   {
2200.     return super.toString() + "["+
2201.            "privilege" + ":" + getPrivilege()+ "]" +
                 System.getProperties().getProperty("line.separator") +
2202.            "  " + "employee = "+(getEmployee()!=null?
                 Integer.toHexString(System.identityHashCode(getEmployee())):"null");
2203.   }
2204. }
2205.
2206.
2207.
```

```
2208. //%% NEW FILE Surgeon BEGINS HERE %%
2209.
2210. /*PLEASE DO NOT EDIT THIS CODE*/
2211. /*This code was generated using the UMPLE 1.29.1.4787.f023c4bb4 modeling
        language!*/
2212.
2213.
2214. import java.util.*;
2215.
2216. /**
2217.  * Specialized doctor who performs advanced procedures on patients.
2218.  */
2219. // line 61 "model.ump"
2220. // line 95 "model.ump"
2221. public class Surgeon extends Doctor
2222. {
2223.
2224.   //------------------------
2225.   // MEMBER VARIABLES
2226.   //------------------------
2227.
2228.   //------------------------
2229.   // CONSTRUCTOR
2230.   //------------------------
2231.
2232.   public Surgeon(String aName, int aId, int aSalary, Hospital aHospital, Ward...
        allWards)
2233.   {
2234.     super(aName, aId, aSalary, aHospital, allWards);
2235.   }
2236.
2237.   //------------------------
2238.   // INTERFACE
2239.   //------------------------
2240.
2241.   public void delete()
2242.   {
2243.     super.delete();
2244.   }
2245.
2246. }
2247.
2248.
2249.
2250. //%% NEW FILE Nurse BEGINS HERE %%
2251.
2252. /*PLEASE DO NOT EDIT THIS CODE*/
2253. /*This code was generated using the UMPLE 1.29.1.4787.f023c4bb4 modeling
        language!*/
2254.
2255.
2256. import java.util.*;
2257.
2258. // line 55 "model.ump"
2259. // line 142 "model.ump"
2260. public class Nurse extends Employee
2261. {
2262.
2263.   //------------------------
2264.   // MEMBER VARIABLES
2265.   //------------------------
```

```
2266.
2267.   //Nurse Associations
2268.   private List<Patient> patients;
2269.
2270.   //------------------------
2271.   // CONSTRUCTOR
2272.   //------------------------
2273.
2274.   public Nurse(String aName, int aId, int aSalary, Hospital aHospital, Ward...
          allWards)
2275.   {
2276.     super(aName, aId, aSalary, aHospital, allWards);
2277.     patients = new ArrayList<Patient>();
2278.   }
2279.
2280.   //------------------------
2281.   // INTERFACE
2282.   //------------------------
2283.   /* Code from template association_GetMany */
2284.   public Patient getPatient(int index)
2285.   {
2286.     Patient aPatient = patients.get(index);
2287.     return aPatient;
2288.   }
2289.
2290.   public List<Patient> getPatients()
2291.   {
2292.     List<Patient> newPatients = Collections.unmodifiableList(patients);
2293.     return newPatients;
2294.   }
2295.
2296.   public int numberOfPatients()
2297.   {
2298.     int number = patients.size();
2299.     return number;
2300.   }
2301.
2302.   public boolean hasPatients()
2303.   {
2304.     boolean has = patients.size() > 0;
2305.     return has;
2306.   }
2307.
2308.   public int indexOfPatient(Patient aPatient)
2309.   {
2310.     int index = patients.indexOf(aPatient);
2311.     return index;
2312.   }
2313.   /* Code from template association_MinimumNumberOfMethod */
2314.   public static int minimumNumberOfPatients()
2315.   {
2316.     return 0;
2317.   }
2318.   /* Code from template association_AddManyToManyMethod */
2319.   public boolean addPatient(Patient aPatient)
2320.   {
2321.     boolean wasAdded = false;
2322.     if (patients.contains(aPatient)) { return false; }
2323.     patients.add(aPatient);
2324.     if (aPatient.indexOfNurse(this) != -1)
2325.     {
```

```
2326.       wasAdded = true;
2327.     }
2328.     else
2329.     {
2330.       wasAdded = aPatient.addNurse(this);
2331.       if (!wasAdded)
2332.       {
2333.         patients.remove(aPatient);
2334.       }
2335.     }
2336.     return wasAdded;
2337.   }
2338.   /* Code from template association_RemoveMany */
2339.   public boolean removePatient(Patient aPatient)
2340.   {
2341.     boolean wasRemoved = false;
2342.     if (!patients.contains(aPatient))
2343.     {
2344.       return wasRemoved;
2345.     }
2346.
2347.     int oldIndex = patients.indexOf(aPatient);
2348.     patients.remove(oldIndex);
2349.     if (aPatient.indexOfNurse(this) == -1)
2350.     {
2351.       wasRemoved = true;
2352.     }
2353.     else
2354.     {
2355.       wasRemoved = aPatient.removeNurse(this);
2356.       if (!wasRemoved)
2357.       {
2358.         patients.add(oldIndex,aPatient);
2359.       }
2360.     }
2361.     return wasRemoved;
2362.   }
2363.   /* Code from template association_AddIndexControlFunctions */
2364.   public boolean addPatientAt(Patient aPatient, int index)
2365.   {
2366.     boolean wasAdded = false;
2367.     if(addPatient(aPatient))
2368.     {
2369.       if(index < 0 ) { index = 0; }
2370.       if(index > numberOfPatients()) { index = numberOfPatients() - 1; }
2371.       patients.remove(aPatient);
2372.       patients.add(index, aPatient);
2373.       wasAdded = true;
2374.     }
2375.     return wasAdded;
2376.   }
2377.
2378.   public boolean addOrMovePatientAt(Patient aPatient, int index)
2379.   {
2380.     boolean wasAdded = false;
2381.     if(patients.contains(aPatient))
2382.     {
2383.       if(index < 0 ) { index = 0; }
2384.       if(index > numberOfPatients()) { index = numberOfPatients() - 1; }
2385.       patients.remove(aPatient);
2386.       patients.add(index, aPatient);
```

```
2387.       wasAdded = true;
2388.     }
2389.     else
2390.     {
2391.       wasAdded = addPatientAt(aPatient, index);
2392.     }
2393.     return wasAdded;
2394.   }
2395.
2396.   public void delete()
2397.   {
2398.     ArrayList<Patient> copyOfPatients = new ArrayList<Patient>(patients);
2399.     patients.clear();
2400.     for(Patient aPatient : copyOfPatients)
2401.     {
2402.       aPatient.removeNurse(this);
2403.     }
2404.     super.delete();
2405.   }
2406.
2407. }
```