Exercise 8.12

a)

We have the following class hierarchy:

Person

- Teacher
- Student

  PhDStudent

1. `Person p1 = new Student();` is valid as Student is also a Person.
2. `Person p2 = new PhDStudent();` is valid as PhDStudent inherits from Student, which inherits from Person.
3. `PhDStudent phd1 = new Student();` is *not* valid because a Student does not inherit from PhD-Student; that is, a student is not automatically a PhDStudent from the given class hierarchy.
4. `Teacher t1 = new Person();` is not valid, as Teacher is a subclass of Person. A Person can also be a Student, i.e. not a Teacher, which is why this declaration is illegal.
5. `Student s1 = new PhDStudent();` is perfectly fine since PhDStudent inherits the Student class.

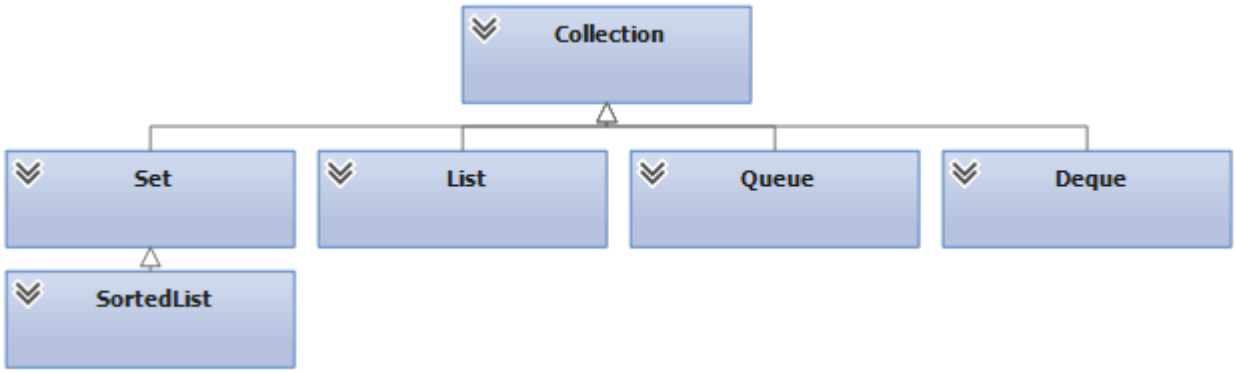
b)

1. `s1 = p1;` is invalid as the Person class does not inherit from Student.
2. `s1 = p2;` is invalid by the same reasoning as above. *p2* is a Person object and as we are trying to assign a Student object a Person object, it will not work.
3. `p1 = s1;` is valid because a Student inherits from a Person. This will, in other words, work!
4. `t1 = s1;` is not valid because there is no explicit connection between a Teacher object and a Student object apart from the fact that they inherit from the same class, that is, Person.
5. `s1 = phd1;` is valid because PhDStudent inherits from a Student.
6. `phd1 = s1;` is invalid because Student does not inherit from a PhD Student.

Exercise 8.14

Nothing has to be changed in the NewsFeed class when another subclass of Post is added because as long as a class inherits from Post, the current code will work. Right now, the addPost method does not explicitly specify the type of Post needed. It accepts all Posts, and thus subclasses of Post class.

Exercise 8.15



Exercise 8.16

LabClass

```
1  import java.util.*;
2
3  /**
4   * The LabClass class represents an enrolment list for one lab class. It
        stores
5   * the time, room and participants of the lab, as well as the instructor's
        name.
6   *
7   * @author Michael Ã¶Klling and David Barnes
8   * @version 2011.07.31
9   */
10 public class LabClass
11 {
12     private Instructor instructor;
13     private String room;
14     private String timeAndDay;
15     private ArrayList<Student> students;
16     private int capacity;
17
18     /**
19      * Create a LabClass with a maximum number of enrolments. All other
            details
20      * are set to default values.
21      */
22     public LabClass(int maxNumberOfStudents)
23     {
24         instructor = null;
25         room = "unknown";
26         timeAndDay = "unknown";
27         students = new ArrayList<Student>();
28         capacity = maxNumberOfStudents;
29     }
30
31     /**
32      * Add a student to this LabClass.
33      */
34     public void enrollStudent(Student newStudent)
35     {
36         if(students.size() == capacity) {
37             System.out.println("The class is full, you cannot enrol.");
38         }
39         else {
40             students.add(newStudent);
41         }
42     }
43
44     /**
45      * Return the number of students currently enrolled in this LabClass.
46      */
47     public int numberOfStudents()
48     {
49         return students.size();
```

```
50      }
51
52      /**
53       * Set the room number for this LabClass.
54       */
55      public void setRoom(String roomNumber)
56      {
57          room = roomNumber;
58      }
59
60      /**
61       * Set the time for this LabClass. The parameter should define the day
62       * and the time of day, such as "Friday, 10am".
63       */
64      public void setTime(String timeAndDayString)
65      {
66          timeAndDay = timeAndDayString;
67      }
68
69      /**
70       * Set the name of the instructor for this LabClass.
71       */
72      public void setInstructor(Instructor instructor)
73      {
74          this.instructor = instructor;
75      }
76
77      /**
78       * Print out a class list with other LabClass details to the standard
79       * terminal.
80       */
81      public void printList()
82      {
83          System.out.println("Lab class " + timeAndDay);
84          System.out.println("Instructor: " + instructor.getCurrentName() +
                "    Room: " + room);
85          System.out.println("Class list:");
86          for(Student student : students) {
87              student.print();
88          }
89          System.out.println("Number of students: " + numberOfStudents());
90      }
91  }
```

Person

```
1
2   /**
3    * Write a description of class Person here.
4    *
5    * @author (your name)
6    * @version (a version number or a date)
7    */
8   public class Person
9   {
10      // instance variables - replace the example below with your own
```

```
11      private String id;
12      private String name;
13      private String email;
14
15      /**
16       * Constructor for objects of class Person
17       */
18      public Person(String id, String name, String email)
19      {
20          // initialise instance variables
21          this.id = id;
22          this.name = name;
23          this.email = email;
24      }
25
26      public String getCurrentName()
27      {
28          return name;
29      }
30
31      public void setName(String name)
32      {
33          this.name = name;
34      }
35
36      public String getID()
37      {
38          return id;
39      }
40  }
```

Student

```
1
2   /**
3    * The Student class represents a student in a student administration
         system.
4    * It holds the student details relevant in our context.
5    *
6    * @author Michael Ã¶Klling and David Barnes
7    * @version 2011.07.31
8    */
9   public class Student extends Person
10  {
11      // the amount of credits for study taken so far
12      private int credits;
13
14      /**
15       * Create a new student with a given name and ID number.
16       */
17      public Student(String fullName, String studentID, String email)
18      {
19          super(studentID, fullName, email);
20          credits = 0;
21      }
22
```

```
23      /**
24       * Return the full name of this student.
25       */
26      public String getName()
27      {
28          return this.getCurrentName();
29      }
30
31      /**
32       * Set a new name for this student.
33       */
34      public void changeName(String replacementName)
35      {
36          this.setName(replacementName);
37      }
38
39      /**
40       * Return the student ID of this student.
41       */
42      public String getStudentID()
43      {
44          return this.getID();
45      }
46
47      /**
48       * Add some credit points to the student's accumulated credits.
49       */
50      public void addCredits(int additionalPoints)
51      {
52          credits += additionalPoints;
53      }
54
55      /**
56       * Return the number of credit points this student has accumulated.
57       */
58      public int getCredits()
59      {
60          return credits;
61      }
62
63      /**
64       * Return the login name of this student. The login name is a
             combination
65       * of the first four characters of the student's name and the first
             three
66       * characters of the student's ID number.
67       */
68      public String getLoginName()
69      {
70          return this.getName().substring(0,4) + this.getID().substring(0,3);
71      }
72
73      /**
74       * Print the student's name and ID number to the output terminal.
75       */
76      public void print()
```

```
77      {
78          System.out.println(this.getName() + ", student ID: " +
                this.getID() + ", credits: " + credits);
79      }
80  }
```

Instructor

```
1
2   /**
3    * Write a description of class Instructor here.
4    *
5    * @author (your name)
6    * @version (a version number or a date)
7    */
8   public class Instructor extends Person
9   {
10      // instance variables - replace the example below with your own
11      private String x;
12
13      /**
14       * Constructor for objects of class Instructor
15       */
16      public Instructor(String id, String name, String email)
17      {
18          // initialise instance variables
19          super(id,name,email);
20      }
21  }
```

Exercise 'exponentiation'

Iterative

The iterative exponentiation algorithm is equivalent to the function α.

$$\alpha(x,n) = \begin{cases} x \times \alpha(x,n-1) & : n \geq 1 \\ 1 & : n = 0 \end{cases}$$

Proof We want to show that this function is equivalent to taking a number $x \in \mathbb{R}$ to the power of a number $n \in \mathbb{N}$. Let P_n denote the case when the exponent is n.

Basis Let's take the case $n = 0$.

$$\begin{aligned} \text{LHS} &= \alpha(n,0) = 1 \\ \text{RHS} &= n^0 = 1 \\ &\because \text{LHS} = \text{RHS} \\ &\therefore P_0 = \text{true} \end{aligned}$$

Induction We assume that P_n is true, i.e. $\alpha(x,n) = x^n, \quad \forall n \forall x$ (*)
Then, for P_{n+1}:

$$\alpha(x,n+1) = x \times \alpha(x,n) = x \times x^n = \text{using (*)} = x^{n+1}$$

$\therefore$ Since P_0 was shown to be true and assuming that P_n is true, it was shown that P_{n+1} is true also. Thus, by the principles of mathematical induction, P_n is true for all $n \in \mathbb{N}$. $\square$

Time complexity The time to execute this function is given by

$$T(n) = \begin{cases} T(n-1)+1 & : n \geq 1 \\ 1 & : n = 0 \end{cases}$$

Sure enough, the closed form of $T(n)$ is:

$$T(n) = n + 1 = O(n) \tag{1}$$

Recursive

The recursive implementation is a bit tricker. In order to prove that the algorithm represents exponentiation, we need to split it into two. The first part, i.e. $\forall n \leq 4$, we know is true because we have shown that the iterative algorithm is true. It need not to be proved. The latter has to be proved, as it is not as clear what is going on. We do know that the last operation (where the recursive call is being made) is true because $\lfloor \frac{n}{2} \rfloor + \lfloor \frac{n+1}{2} \rfloor = n$, so the exponent is not altered. In other words, the recursive function will keep breaking up the exponent (without changing the value) until the exponent is less than or equal to 4, when the iterative algorithm is called.

The time complexity for the recursive algorithm is given by (not including the iterative method call):

$$T(n) = \begin{cases} 2T(n/2)+1 & : n \geq 1 \\ 1 & : n = 0 \end{cases}$$

That is because it keeps breaking up the problem into two. The reason why the time complexity of all method calls to the iterative procedure were omitted is because they contribute with a constant number of operations. Given $n \implies T(n) = n + 1$. Thus, $O(n)$, as we already have shown. Using Master Theorem, since $0 < \log_2 2 = 1$, the time complexity of the recursive algorithm is also $O(n)$.