# DD1339 Introduktion till datalogi 2013/2014

# Uppgift nummer: 1

# Namn: Marcus Larsson

# Grupp nummer: 5

# Övningsledare: Marcus Dicander

# Betyg: ..... Datum: .............. Rättad av: .......................................

# Exercise 8.12

## a.

### Giltig kod:

Person p1 = new Student();

Person p2 = new PhDStudent();

Student s1 = new PhDStudent();

Eftersom de klasser till vänster, den variabeln som tilldelas är moderklass till den klassen som tilldelas varabeln

### Ej giltig:

PhDStudent phd1 = new Student();

Teacher t1 = new Person();

Precis tvärtom av ovan.

Man kan tänka såhär: Student är en Person men en Person är inte nödvändigtvis en Student eftersom det kan också vara Teacher. Därför är dem 2 sista kodraderna ovan ej giltiga.

## b.

### Giltiga:

p1=s1;

s1=phd1;

### Ej giltiga:

s1=p1;

s1=p2;

t1=s1;

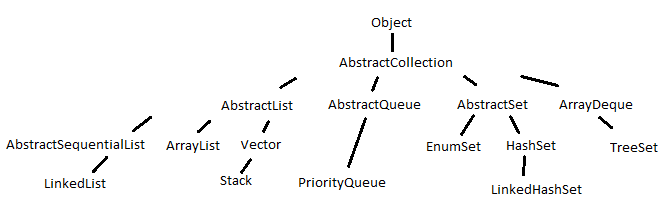
phd1=s1;

Samma motivering som i a. Man har bara sagt att s1 är en Student och p1 och p2 är en Person.

# Exercise 8.14

Inget. Eftersom NewsFeed bara använder sig av klassen Post så är en ny sublass av Post också en ett Post-objekt. Genom polymorfism berör det inte NewsFeed.

# Exercise 8.15



# Exercise 8.16

## LabClass

import java.util.\*;

/\*\*

\* The LabClass class represents an enrolment list for one lab class. It stores

\* the time, room and participants of the lab, as well as the instructor's name.

\*

\* @author Michael Kölling and David Barnes

\* @version 2011.07.31

\*/

public class LabClass

{

private Instructor instructor;

private String room;

private String timeAndDay;

private ArrayList<Student> students;

private int capacity;

/\*\*

\* Create a LabClass with a maximum number of enrolments. All other details

\* are set to default values.

\*/

public LabClass(int maxNumberOfStudents)

{

instructor = new Instructor("Marcus Dicander", "666");

room = "unknown";

timeAndDay = "unknown";

students = new ArrayList<Student>();

capacity = maxNumberOfStudents;

}

/\*\*

\* Add a student to this LabClass.

\*/

public void enrollStudent(Student newStudent)

{

if(students.size() == capacity) {

System.out.println("The class is full, you cannot enrol.");

}

else {

students.add(newStudent);

}

}

/\*\*

\* Return the number of students currently enrolled in this LabClass.

\*/

public int numberOfStudents()

{

return students.size();

}

/\*\*

\* Set the room number for this LabClass.

\*/

public void setRoom(String roomNumber)

{

room = roomNumber;

}

/\*\*

\* Set the time for this LabClass. The parameter should define the day

\* and the time of day, such as "Friday, 10am".

\*/

public void setTime(String timeAndDayString)

{

timeAndDay = timeAndDayString;

}

/\*\*

\* Set the name of the instructor for this LabClass.

\*/

public void setInstructor(Instructor instructor)

{

this.instructor = instructor;

}

/\*\*

\* Print out a class list with other LabClass details to the standard

\* terminal.

\*/

public void printList()

{

System.out.println("Lab class " + timeAndDay);

System.out.println("Instructor: " + instructor + " Room: " + room);

System.out.println("Class list:");

for(Student student : students) {

student.print();

}

System.out.println("Number of students: " + numberOfStudents());

}

}

## Person

/\*\*

\* A person is in this project a student or instructor. This class is abstract so objects of this class needs to be an Instroctor or Student.

\*

\* @author (Marcus Larsson)

\* @version 2014-01-10

\*/

public abstract class Person

{

// the person's full name

protected String name;

// the person's ID

protected String id;

/\*\*

\* Create a new Person with a given name and ID number.

\*/

public Person(String fullName, String studentID)

{

name = fullName;

id = studentID;

}

/\*\*

\* Return the full name of this person.

\*/

public String getName()

{

return name;

}

/\*\*

\* Set a new name for this person.

\*/

public void changeName(String replacementName)

{

name = replacementName;

}

/\*\*

\* Return the ID of this person.

\*/

public String getID()

{

return id;

}

/\*\*

\* Return the login name of this person. The login name is a combination

\* of the first four characters of the student's name and the first three

\* characters of the person's ID number.

\*/

public String getLoginName()

{

return name.substring(0,4) + id.substring(0,3);

}

/\*\*

\* @return Returns a String with the persons name and id.

\*/

public String toString(){

return name+", ID: " + id;

}

}

## Instructor

/\*\*

\* An instructor is a teacher for a class. So far no extra methods than the ones from Person.

\*

\* @author Marcus Larsson

\* @version 2014-01-10

\*/

public class Instructor extends Person

{

/\*\*

\* Constructor for objects of class Instructor

\*/

public Instructor(String fullName, String instructorID)

{

super(fullName, instructorID);

}

}

## Student

/\*\*

\* The Student class represents a student in a student administration system.

\* It holds the student details relevant in our context.

\*

\* @author Michael Kölling and David Barnes, modified by Marcus Larsson

\* @version 2014.01.10

\*/

public class Student extends Person

{

// the amount of credits for study taken so far

private int credits;

/\*\*

\* Create a new student with a given name and ID number.

\*/

public Student(String fullName, String studentID)

{

super(fullName, studentID);

credits = 0;

}

/\*\*

\* Add some credit points to the student's accumulated credits.

\*/

public void addCredits(int additionalPoints)

{

credits += additionalPoints;

}

/\*\*

\* Return the number of credit points this student has accumulated.

\*/

public int getCredits()

{

return credits;

}

/\*\*

\* Print the student's name and ID number to the output terminal.

\*/

public void print()

{

System.out.println(name + ", student ID: " + id + ", credits: " + credits);

}

}

# Exercise web

double expIterativ(double x, int n) {

double res = 1.0;

for (int i = 0; i < n; i++)

res \*= x;

return res;

}

double expRekursiv(double x, int n) {

if (n <= 4)

return expIterativ(x, n);

return expRekursiv(x, n/2) \*

expRekursiv(x, (n + 1)/2);

}

## Bevisa iterativ

För bevis med en invariant av den iterativa loopen ska vi visa att 3 villkor är uppfyllda. Invarianten är variablen res som är 1.0\*x^i.

1. Initiering

Initieringen är sann eftersom vi sätter res = 1.0 i början. Innan loopen utförs så är då res = 1.0\*x^0 vilket är 1. Då¨x är ett flyttal. Om x blir det konstigt då 0^0 ej är definierat och då svarar programmet 1. Men om vi har x!=0 så gäller det för alla x. Vi kan också se att om x=0 och n>0 så gäller invarianten. Det blir bara problem när båda är 0.

2. Uppdatering

Om invarianten är sann för en iteration så är den även det vid nästa. Eftersom res = 1.0\*x^i så blir det vid första iterationen res=1.0\*x^1=x^1 vilket stämmer. Det är därmed bevisat.

3. Avslutning

När loopen avslutas ska invarianten fortfarande gälla. Sista iterationen blir res=x^(i)\*x = x^(i+1). Sedan ökar i med 1. Då evalueras villkoret i<n till falskt eftersom n är ett heltal är då i=n sista gången. Invarianten ger då x^i = x^n och därmed är avslutningen också sann.

## Bevisa rekursiv

För den rekursiva metoden så säger den att kalla på den iterativa om n<=4. Då har vi i förra beviset visat att p(i) är sant för i<5. Vi har alltså ett basfall där p(i) är sant för 0<i<=4. Vi vill nu visa att p(5) också är sant först.

Vid n=5 anropas den andra delen av rekursiva metoden och vi får return (expRekursiv(x, 2)\*expRekursiv(x, 3). Eftersom 2<3<4 anropas iterativa som ger att uttrycket blir x^2\*x^3=x^5 som vi ville ha. p(5) stämmer alltså.

Nu har vi vår rekursiva basfall. Nu ska vi bevisa att p(k) stämmer för alla k>=6.

Induktionsantagandet p(i) är sant för alla i<k ger att p(k)=p(k-1)\*x där antagandet säger att p(k-1) = x^(k-1). Det ger att p(k-1)\*k = x^(k-1)\*k=x^k vilket är det sanna svaret och därmed är det bevisat.

## Tidskomplexitet

### Iterativa metoden

Jag väljer tilldelningen res\*=x som karakteristisk operation. Den utförs från i=0 till i<n. Alltså n gånger. Tidskomplexitet blir då O(n).

### Rekursiva metoden

Vi kan räkna if-satsen som en operation vid varje anrop av den rekursiva metoden som vi kan kalla T.

T(n) är då lika med 1 för n<=4. (Den iterativa metoden körs O(4))

För n>4 har vi T(n) = 1 + T(n/2) + T((n+1)/2). Eftersom n>4 är (n+1/2) nästan samma som n/2 och då n går mot oändligheten kan vi bortse från +1. Vi har då att T(n) ->1+2T(n/2) = O(n) för stora n.

Tidskomplexiteten är alltså O(n).