Object-Orientated Programming

CMM24

RGU (2023)

Session 7 (Object Orientated Programming)

Introduction to Inheritance:

Extends

Instanceof

CMM024

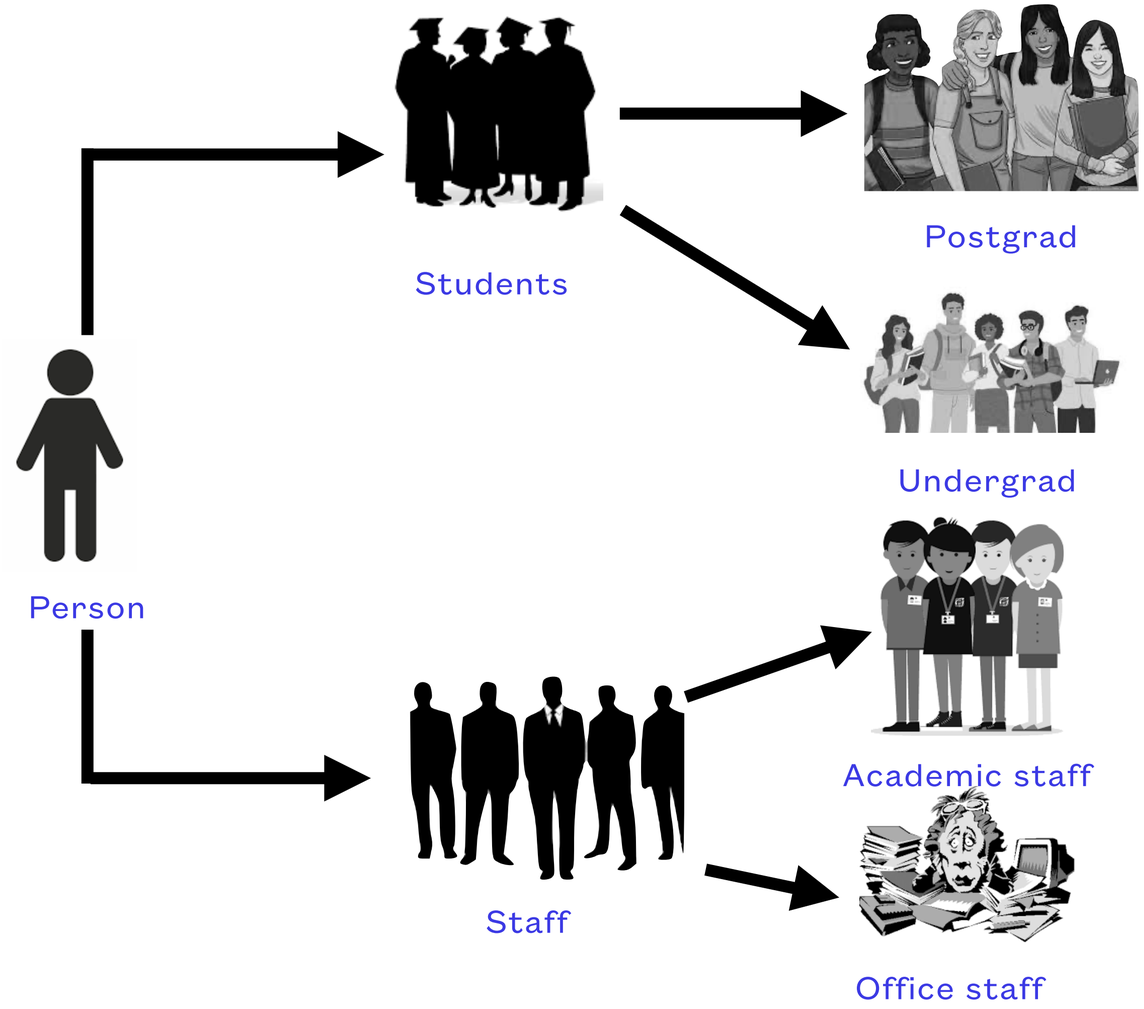
Summary

- **Discuss what is a inheritance**
- **Discuss how to model inheritance**
- **Go through an example by modelling staff PT or FT**
- **Testing a class**

What we learnt so Far

- Last week we have learnt about what are classes and how we can model almost anything in an object orientated manner
- We also learnt about UML to visually represent object orientated concepts
- It is time to go deeper into this paradigm and to examine the concept of inheritance in object orientated programming

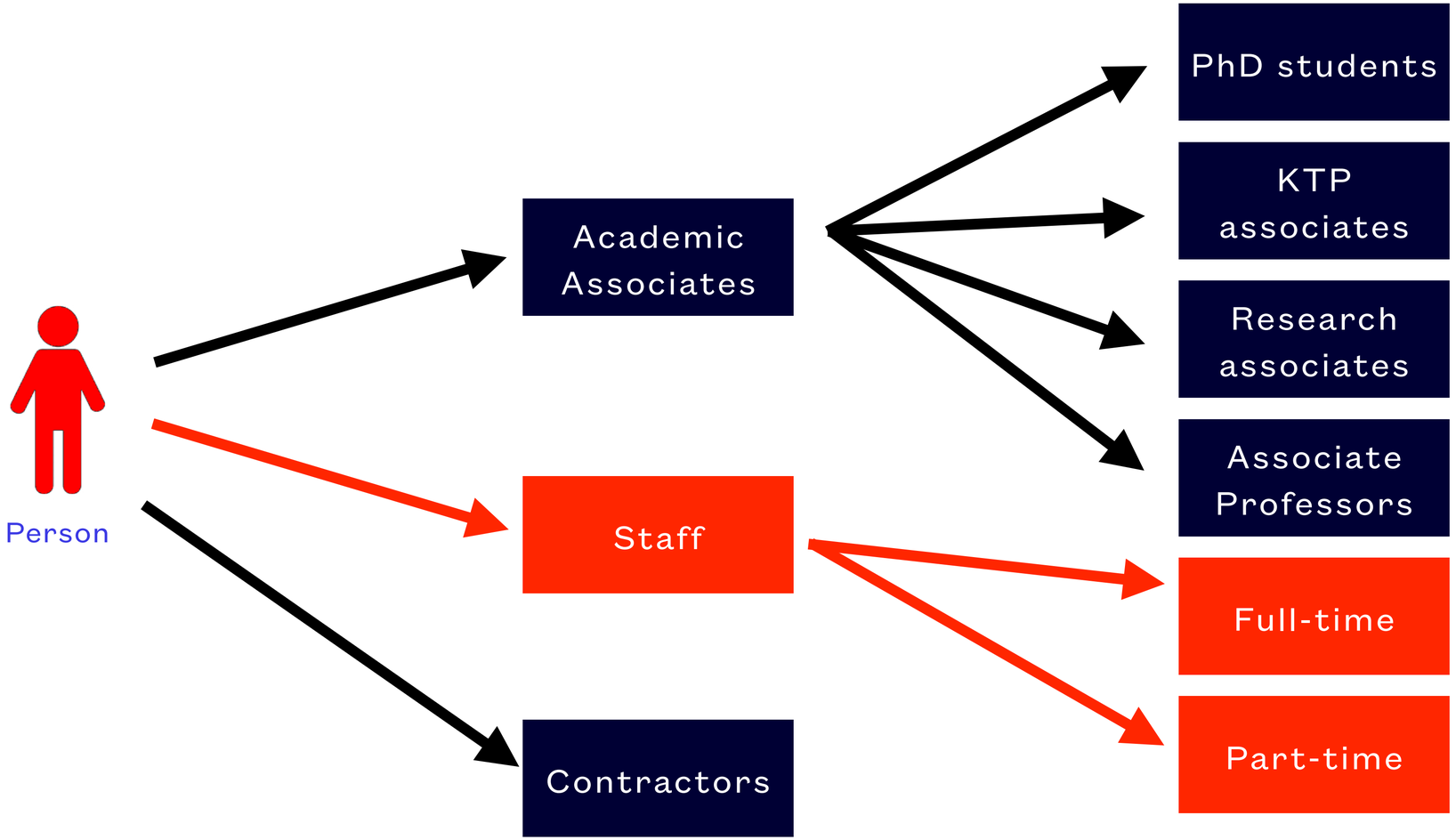
How class relate to each other?



Inheritance

- We can see that there are many relationships between these types of people
  - Students are persons but there is something special about them as he or she are registered to the university to study, thus have a student ID, a list of modules to attend for a particular school. They can be undergrad or postgrad, thus have different level of requirements, and study will only be one year for postgrad etc...
  - Staff are persons too. They can either work in the office and run the school or be academic staff whose main purpose is essentially teaching, research or both
- These relationships is the basis of what we know as inheritance

How class relate to each other?



public Person

private name: String

private address: String

private age: int

private Person(name: String, address String, age: int)

public toString():String

public Staff

private name: String

private address: String

private age: int

private staffID: String

private grade: int

private school: String

private NI: String

private Staff(name: String, address String, age: int, staffID: String, grade: int, school: string, NI: String)

public toString():String

public FTStaff

private name: String

private address: String

private age: int

private staffID: String

private grade: int

private school: String

private NI: String

private YearlySalary: double

private FTStaff(name: String, address: String, age: int, StaffID: String, grade: int, school: String, NI: String, YearlySalary: double)

public toString():String

public PTStaff

private name: String

private address: String

private age: int

private staffID: String

private grade: int

private school: String

private NI: String

private hourlyRate: double

private numberHours: double

private PTStaff(name: String, address: String, age: int, StaffID: String, grade: int, school: String, NI: String, hourlyRate: double, numberHours: double)

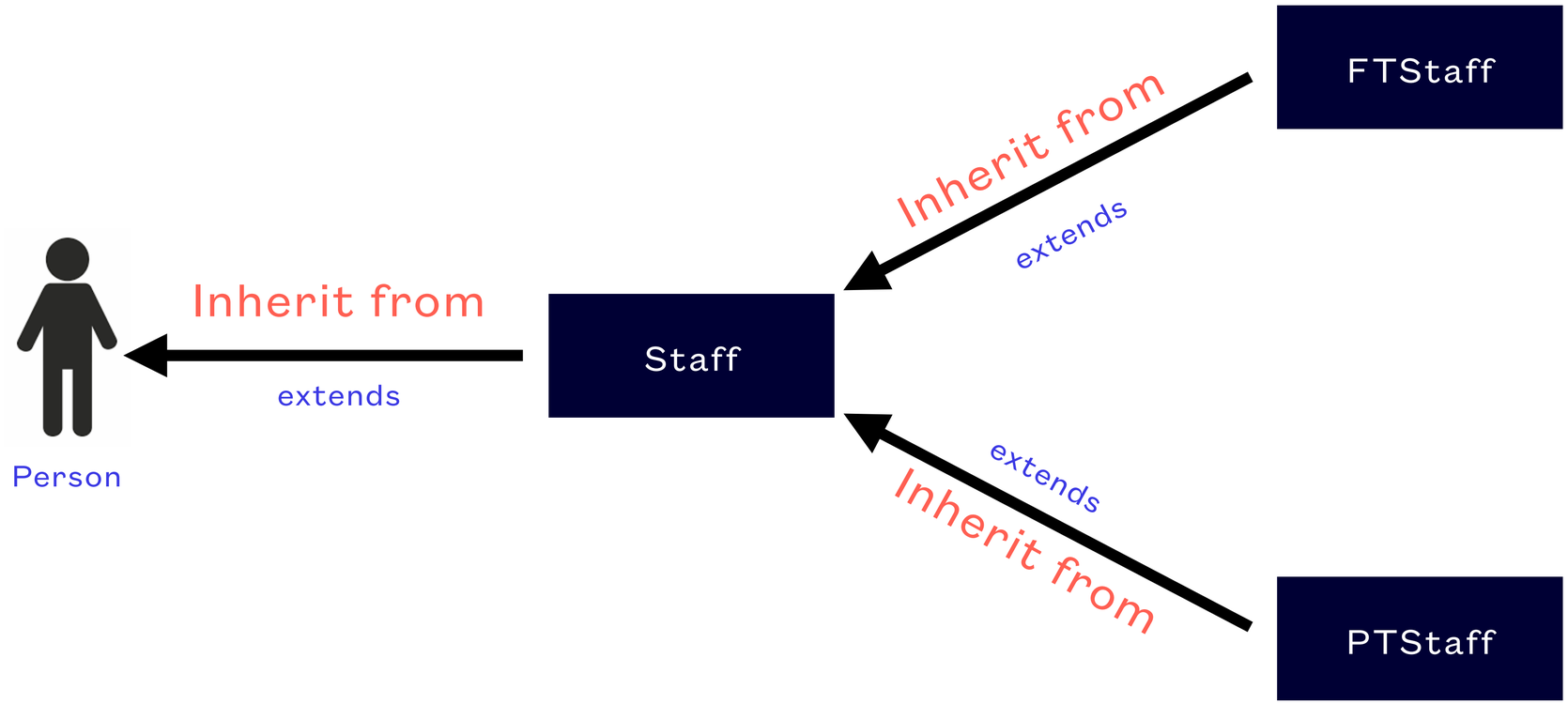
public toString():String

Classes as they are

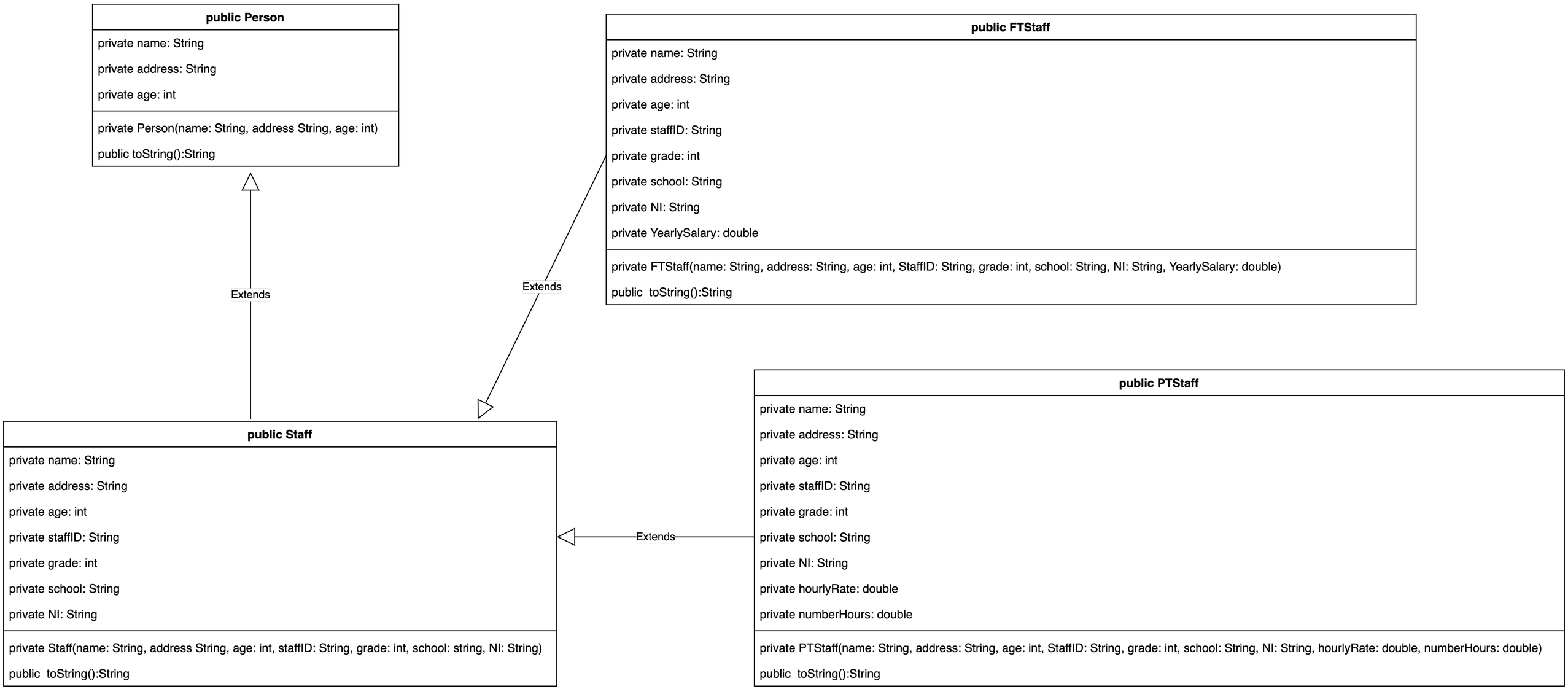
Why using Inheritance

- We have seen in the last slide that there is a huge amount of repetitions
  - All the red fields below to the Person class
  - All the green fields belong to the Staff class
  - Only the blue fields are specific to particular employee
- We stop these repetitions by using the principle of Inheritance
  - A staff is a Person
  - A Full-time staff is a Staff but also a Person
  - A Part-time staff is a Staff but also a Person

A class extends another

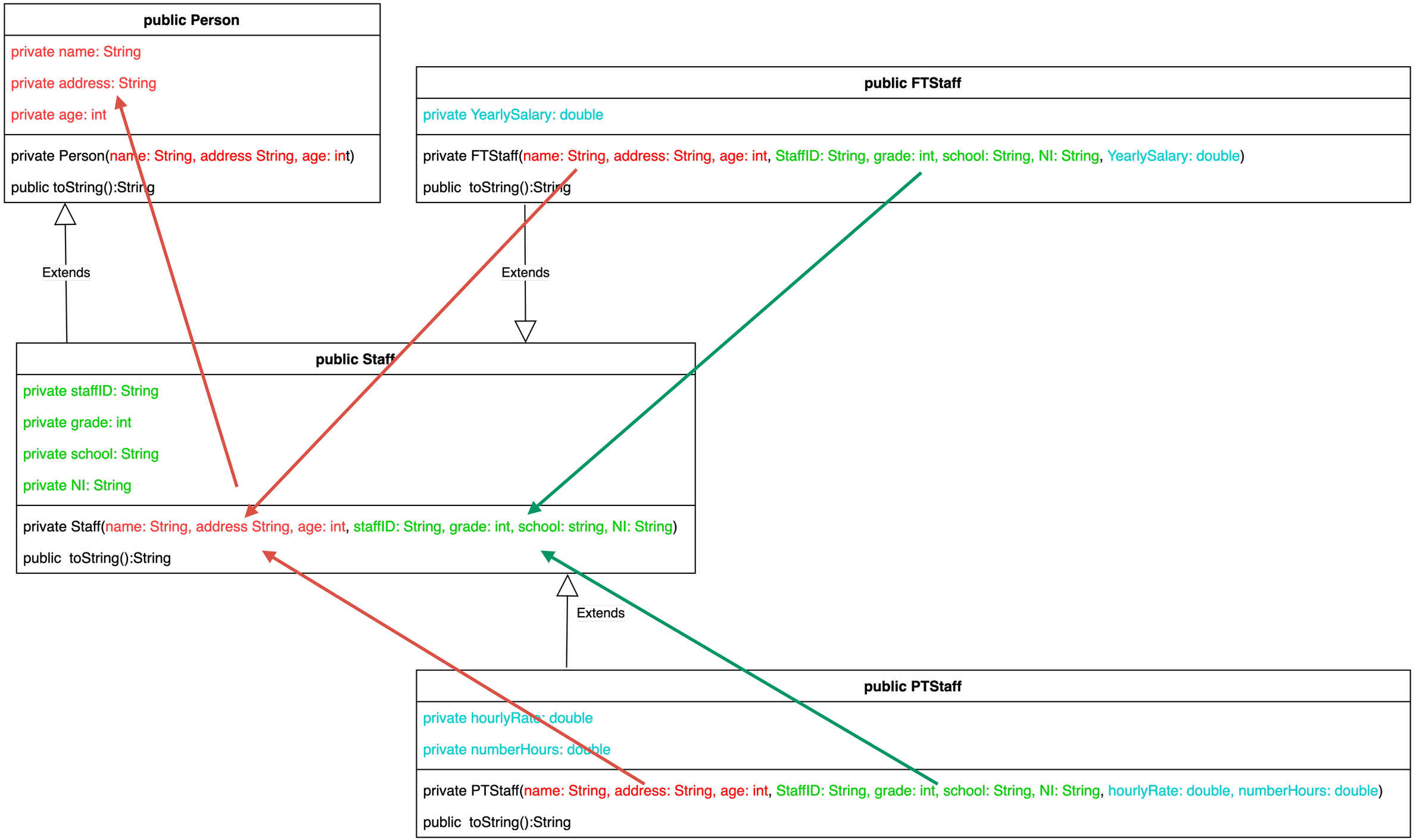


- A class inherits the properties of another class by extending a relationship from a class to another

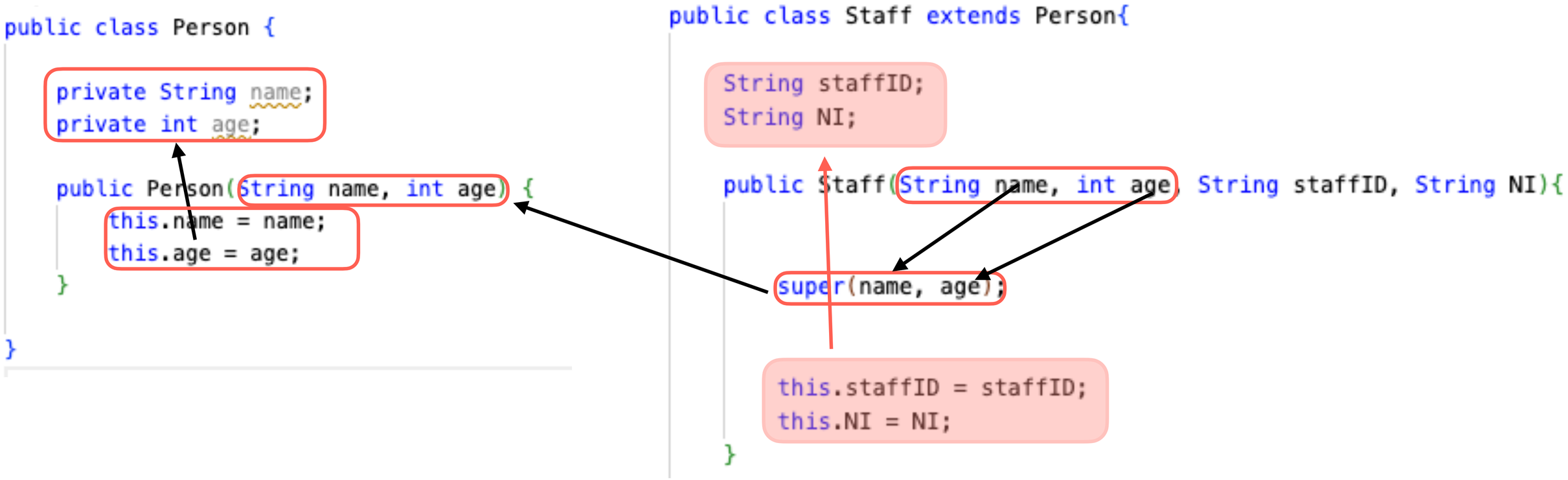


Classes as they are

- When a class inherits from another class:
  - The base class is called a superclass (or base class)
  - The class that inherits the properties of a superclass is called a subclass
  - The subclass when creating an object instance passes the properties that belong to superclass in the constructor but does not retain them.
  - This is done using the super keyword used in the first line in the constructor


public Person
private name: String
private address: String
private age: int
private Person(name: String, address String, age: int)
public toString():String
public FTStaff
private YearlySalary: double
private FTStaff(name: String, address: String, age: int, StaffID: String, grade: int, school: String, NI: String, YearlySalary: double)
public toString():String
Extends
Extends
public Staff
private staffID: String
private grade: int
private school: String
private NI: String
private Staff(name: String, address String, age: int, staffID: String, grade: int, school: string, NI: String)
public toString():String
Extends
public PTStaff
private hourlyRate: double
private numberHours: double
private PTStaff(name: String, address: String, age: int, StaffID: String, grade: int, school: String, NI: String, hourlyRate: double, numberHours: double)
public toString():String

How inheritance works



The line super(name, age) is in fact invoking the superclass constructor

How inheritance works

```
public class Staff extends Person{

    String staffID;
    String NI;

    public Staff(String name, int age, String staffID, String NI){

        super(name, age);

        this.staffID = staffID;
        this.NI = NI;
    }
```

```
public class Person {
    private String name;
    private int age;
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

```
public class FTStaff extends Staff {

    double yearlyWage;

    public FTStaff(String name, int age, String staffID, String NI, double yearlyWage) {

        super(name, age, staffID, NI);

        this.yearlyWage = yearlyWage;
    }
}
```

How inheritance works

```
public class Staff extends Person{

    String staffID;
    String NI;

    public Staff(String name, int age, String staffID, String NI){

        super(name, age);

        this.staffID = staffID;
        this.NI = NI;
    }
```

```
public class Person {
    private String name;
    private int age;
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

```
public class PTStaff extends Staff {

    double hourlyWage;
    double numberHours;

    public PTStaff(String name, int age, String staffID, String NI, double hourlyWage, double numberHours) {

        super(name, age, staffID, NI);

        this.hourlyWage = hourlyWage;
        this.numberHours = numberHours;
    }
}
```

```java
public class Person {
    private String name;
    private int age;
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

```java
public class Staff extends Person{
    String staffID;
    String NI;
    public Staff(String name, int age, String staffID, String NI){
        super(name, age);
        this.staffID = staffID;
        this.NI = NI;
    }
}
```


```java
public class PTStaff extends Staff {
    double hourlyWage;
    double numberHours;
    public PTStaff(String name, int age, String staffID, String NI, double hourlyWage, double numberHours) {
        super(name, age, staffID, NI);
        this.hourlyWage = hourlyWage;
        this.numberHours = numberHours;
    }
}
```

```java
public class FTStaff extends Staff {
    double yearlyWage;
    public FTStaff(String name, int age, String staffID, String NI, double yearlyWage) {
        super(name, age, staffID, NI);
        this.yearlyWage = yearlyWage;
    }
}
```

Remember: you must have enough parameters in the top subclass constructor to initialise all the subclasses fields!

Adding functionality

- We can add functionality for the Part-time and full-time employees
- If you are a full-time employee, you get an annual salary
- If you are a part-time employee, your salary is the product of the number of hours by your hourly rate
- We can program the toString method

```
public class FTStaff extends Staff {
    double yearlyWage;
    public FTStaff(String name, int age, String staffID, String NI, double yearlyWage) {
        super(name, age, staffID, NI);
        this.yearlyWage = yearlyWage;
    }
    public double getSalary(){
        return yearlyWage;
    }
    public String toString(){
        String st = "\n";
        st += "Yearly Salary: " + this.yearlyWage;
        st += "\n";
        return st;
    }
}

public class PTStaff extends Staff {
    double hourlyWage;
    double numberHours;
    public PTStaff(String name, int age, String staffID, String NI, double hourlyWage, double numberHours) {
        super(name, age, staffID, NI);
        this.hourlyWage = hourlyWage;
        this.numberHours = numberHours;
    }
    public double getHourlyWage(){
        return hourlyWage;
    }
    public double getHoursWorked(){
        return numberHours;
    }
    public String toString(){
        String st = "\n";
        st += "Hourly Wage: " + this.hourlyWage;
        st += "Hours: " + this.numberHours;
        st += "\n";
        return st;
    }
}
```

Adding functionality

- We can also create an Employees database class to store and display details of all employees.
- Here we use 2 distinct collections to store our FT and PT employees
- BUT THERE IS SOMETHING MISSING!
- Remember that a PTStaff is a Staff and therefore inherits data and functionality of Staff. Staff is also a Person (extends Person) thus it inherits data and functionality of Person. Thus a PTStaff, inherits both Staff and Person data and functionality

```
mport java.util.ArrayList;

public class Employees {

    ArrayList<PTStaff> ptEmployees;
    ArrayList<FTStaff> ftEmployees;

    public Employees() {
        ptEmployees = new ArrayList<>();
        ftEmployees = new ArrayList<>();
    }
    public void addPTStaff(PTStaff ptStaff) {
        ptEmployees.add(ptStaff);
    }
    public void addFTStaff(FTStaff ftStaff) {
        ftEmployees.add(ftStaff);
    }
    public void listAllEmployees() {
        for (PTStaff ptStaff : ptEmployees) {
            System.out.println(ptStaff.toString());
        }
        for (FTStaff ftStaff : ftEmployees) {
            System.out.println(ftStaff.toString());
        }
    }
    public static void main(String[] args) {

        PTStaff ptStaff1 = new PTStaff("JCG", 63, "1234", "NP114589N", 20.5, 120);
        PTStaff ptStaff2 = new PTStaff("MAX", 63, "4567", "NP194599N", 17.75, 80);

        FTStaff ftStaff1 = new FTStaff("MILES", 63, "8901", "NP122599N", 34000);
        FTStaff ftStaff2 = new FTStaff("STEPH", 63, "2345", "NP294879N", 29000);

        Employees employees = new Employees();

        employees.addPTStaff(ptStaff1);
        employees.addPTStaff(ptStaff2);
        employees.addFTStaff(ftStaff1);
        employees.addFTStaff(ftStaff2);

        employees.listAllEmployees();
    }
}
```

Adding functionality

- So in order to be able to retrieve some relevant information about the person who is a member of staff, being either a PT or FT.....
- We must add some getter methods in Person to retrieve the name of the staff, and in Staff to retrieve the StaffID of a staff for example

```
public class Staff extends Person{
    String staffID;
    String NI;
    public Staff(String name, int age, String staffID, String NI){
        super(name, age);
        this.staffID = staffID;
        this.NI = NI;
    }
    public String getStaffID(){
        return staffID;
    }
}

 public class Person {
     private String name;
     private int age;
     public Person(String name, int age) {
         this.name = name;
         this.age = age;
     }
     public String getName(){
         return name;
     }
 }
```

Using superclass inherited functionality

- And re-code the Employees class to use these getter methods into the listing of the employees
- We can see that inheritance allows a subclass to access all the superclasses data etc.
- Even if the data and functionality is well down the inheritance chain!
- This is very useful and stops repetition of code

```
public void listAllEmployees() {
    for (PTStaff ptStaff : ptEmployees) {
       String st = "";
        st += "Name: " + ptStaff.getName() + "\t";;
        st += "Staff ID: " + ptStaff.getStaffID();
        st += ptStaff.toString();
        System.out.println(st);
    }
    for (FTStaff ftStaff : ftEmployees) {
       String st = "";
       st += "Name: " + ftStaff.getName() + "\t";
       st += "Staff ID: " + ftStaff.getStaffID();
       st += ftStaff.toString();
       System.out.println(st);
    }
}
```

```
_ws/Session6LectureStuff_c91e1e73/b
Name: JCG       Staff ID: 1234
Hourly Wage: 20.5Hours: 120.0

Name: MAX       Staff ID: 4567
Hourly Wage: 17.75Hours: 80.0

Name: MILES     Staff ID: 8901
Yearly Salary: 34000.0

Name: STEPH     Staff ID: 2345
Yearly Salary: 29000.0
```

Using inheritance

- There is a problem
- We are storing PT and FT employees in different ArrayLists
- This creates complexity that we do not want
- However both PT and FT employees are Staff. So we can store the employees not as PTStaff or FTStaff but as Staff, thus using only one ArrayList
- Hence when listing we only need to use one for loop.

```
import java.util.ArrayList;

public class Employees {
    ArrayList<PTStaff> ptEmployees;
    ArrayList<FTStaff> ftEmployees;
    public Employees() {…
    public void addPTStaff(PTStaff ptStaff) {
        ptEmployees.add(ptStaff);
    }
    public void addFTStaff(FTStaff ftStaff) {
        ftEmployees.add(ftStaff);
    }
    public void listAllEmployees() {…
    Run | Debug
    public static void main(String[] args) {…
}
```

Using the superclass type to store employees

- The code becomes very simple
- But what if I want to invoke specific methods of PTStaff (hours & hourly wage) without using to toString() method or for FTStaff we are in trouble!

```
import java.util.ArrayList;
public class Employees {
    ArrayList<Staff> employees;
    public Employees() {
        employees = new ArrayList<>();
    }
    public void addStaff(Staff staff) {
        employees.add(staff);
    }
    public void listAllEmployees() {
        for (Staff staff : employees) {
            String st = "";
            st += "Name: " + staff.getName() + "\t";;
            st += "Staff ID: " + staff.getStaffID();
            st += staff.toString();
            System.out.println(st);
        }
    }
    public static void main(String[] args) {
        PTStaff ptStaff1 = new PTStaff("JCG", 63, "1234", "NP114589N", 20.5, 120);
        PTStaff ptStaff2 = new PTStaff("MAX", 63, "4567", "NP194599N", 17.75, 80);
        FTStaff ftStaff1 = new FTStaff("MILES", 63, "8901", "NP122599N", 34000);
        FTStaff ftStaff2 = new FTStaff("STEPH", 63, "2345", "NP294879N", 29000);
        Employees employees = new Employees();
        employees.addStaff(ptStaff1);
        employees.addStaff(ptStaff2);
        employees.addStaff(ftStaff1);
        employees.addStaff(ftStaff2);
        employees.listAllEmployees();
    }
}
```

Instance of

- We cannot as the employees store Staff objects but only PTStaff and FTStaff know about salary!
- To solve the problem we need to find if the staff is PT or FT, then we can cast the staff to the proper class, and then invoke the right functionality

```
public void listAllEmployees() {
    for (Staff staff : employees) {
        String st =
        st += "Name:
        st += "Staff
        st += staff.getSalary();
        st += staff.toString();
        System.out.println(st);
    }
```

The method getSalary() is undefined for the type Staff Java(67108964)

View Problem (⌥F8) Quick Fix... (⌘.)

Instance of

- We cannot as the employees store Staff objects but only PTStaff and FTStaff know about salary!
- Remember that Stand are really PTStaff and FTStaff
- To solve the problem we need to find if the staff is PT or FT, cast the staff to the proper class, and then invoke the right functionality
- Then we can typecast Staff into its rightful type
- We use the instanceof keyword

```
public void listAllEmployees() {
    for (Staff staff : employees) {
        String st = "";
        st += "Name: " + staff.getName() + "\t";
        st += "Staff ID: " + staff.getStaffID() + "\n";
        if (staff instanceof PTStaff) {
            PTStaff ptStaff = (PTStaff)staff;
            st += "PT - ";
            st += "\tHourly Wage: " + ptStaff.getHourlyWage();
            st += "\tHours Worked: " + ptStaff.getHoursWorked();
            st += "\tYearly Salary: " + ptStaff.getHourlyWage() *  ptStaff.getHoursWorked();
        }else if (staff instanceof FTStaff){
            FTStaff ftStaff = (FTStaff)staff;
            st += "FT - ";
            st += "\tYearly Salary: " + ftStaff.getSalary();
        }
        System.out.println(st);
    }
}
```

```
Name: JCG       Staff ID: 1234
PT -    Hourly Wage: 20.5       Hours Worked: 120.0     Yearly Salary: 2460.0
Name: MAX       Staff ID: 4567
PT -    Hourly Wage: 17.75      Hours Worked: 80.0      Yearly Salary: 1420.0
Name: MILES     Staff ID: 8901
FT -    Yearly Salary: 34000.0
Name: STEPH     Staff ID: 2345
FT -    Yearly Salary: 29000.0
jean-claude@MacBook-Pro-2 Session6LectureStuff %
```

Next week

- **Introduction to abstract classes**
- **Introduction to Enum**