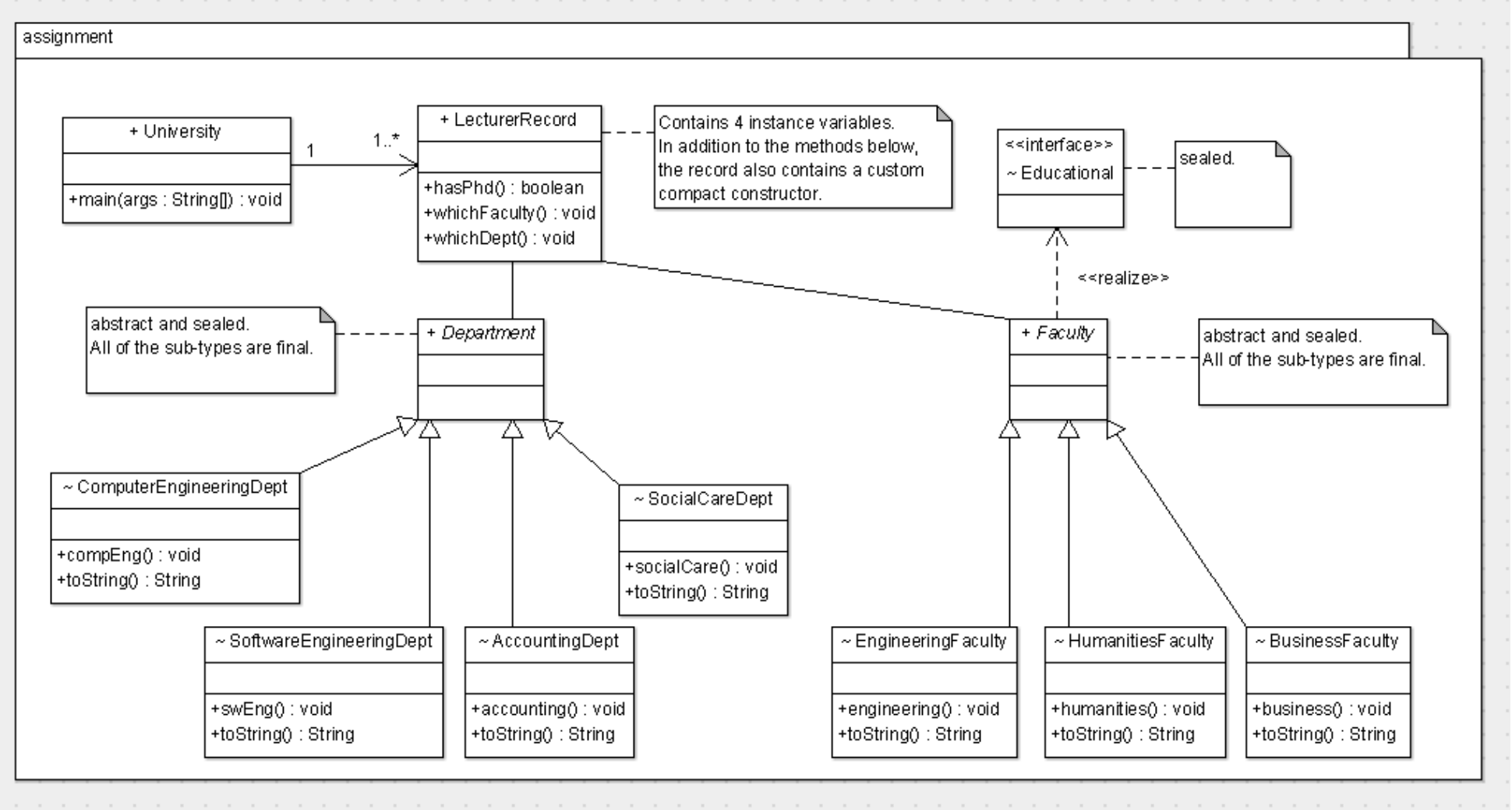
Java 17 Assignment Instructions



Note: As this is <u>not</u> a UML course, the above diagram is an overview. Also, for those of you for which English is not you first language and/or you are not familiar with Universities, a Faculty contains several Departments.

The following instructions will help in coding the assignment:

- the package name is *assignment*
- the *Department* type and all its subtypes are in the *Department.java* file
    - *Department* is *public*; all its subtypes are package-private
    - *Department* is *abstract* and ***sealed***; its subtypes are *final*
    - *ComputerEngineeringDept*:
        - *compEng()* method: output "Custom computer engineering" (a simple tracer message)
        - *toString()* method: returns "Computer Engineering"
    - *SoftwareEngineeringDept*:
        - *swEng()* method: output "Custom software engineering" (a simple tracer message)
        - *toString()* method: returns "Software Engineering"
    - *SocialCareDept*:
        - *socialCare()* method: output "Custom social care" (a simple tracer message)
        - *toString()* method: returns "Social Care"
    - *AccountingDept*:
        - *accounting()* method: output "Custom accounting" (a simple tracer message)
        - *toString()* method: returns "Accounting"
- the *Faculty* type and all its subtypes are in the *Faculty.java* file
    - *Faculty* implements the interface *Educational*
        - *Educational* is a marker interface (no methods) and is *sealed* and package-private
    - *Faculty* is *public*; all its subtypes are package-private
    - *Faculty* is *abstract* and ***sealed***; its subtypes are *final*
    - *EngineeringFaculty*:
        - *engineering()* method: output "We teach computer science, civil engineering etc…"
        - *toString()* method: returns "Engineering"
    - *HumanitiesFaculty*:
        - *humanities()* method: output "We teach social care, European studies etc…"
        - *toString()* method: returns "Humanties"
    - *BusinessFaculty*:
        - *business()* method: output "We teach accountancy, law, economics etc…"
        - *toString()* method: returns "Business"

- *LecturerRecord*
  - *final* by default (insert *final* anyway)
  - the **record** contains 4 instance variables in this order:
    - a *String name*
    - an *Integer age*
    - a *Faculty* reference called *faculty*
    - a *Department* reference called *dept*
  - insert a custom compact constructor:
    - if the *name* passed in is blank (hint: use the *isBlank()* method) or the *age* passed in negative we will throw an *IllegalArgumentException* with a custom message. The custom message is built as follows:
      - using a **text block** we can use the *String* method *formatted* to insert both the *name* and *age* parameters into the custom error message; this can be done as follows:

```
// String interpolation
String errorMsg = """
        Illegal argument passed:
              "name": %s,
              "age": %s
         """.formatted(name, age);
```

  - *hasPhd()* method:
    - we need to cater for someone using "Dr." at the start of their name or "PhD" at the end of their name.
    - figure out the prefix (first 3 characters) and suffix (last 3 characters) in the name. Hint: one option is to use the *substring()* method from *String*
    - using nested **switch expressions**, return *true* if the lecturers name begins with "Dr." or ends with "PhD"; return *false* otherwise.
  - *whichFaculty()* method:
    - switch on the faculty:
      - using a **switch expression pattern matching**:
        - if it's the *EngineeringFaculty* then in a code block do the following:
          - call *toString()* on the reference, prepended with "Faculty of: " i.e. assuming *eng* is the reference, code *System.out.println("Faculty of: "+eng);*
          - call the custom method *engineering()*
        - if it's the *HumanitiesFaculty* then in a code block do the following:
          - call *toString()* on the reference prepended with "Faculty of: "
          - call the custom method *humanities()*
        - if it's the *BusinessFaculty* then in a code block do the following:
          - call *toString()* on the reference prepended with "Faculty of: "
          - call the custom method *business()*
        - otherwise, throw an *IllegalArgumentException*, outputting the faculty that is causing the error in the error message.

- *whichDept()* method:
    - switch on the department:
        - using a **switch expression pattern matching**:
            - if it's the *ComputerEngineeringDept* then in a code block do the following:
                - call *toString()* on the reference prepended with "Dept of: "
                - call the custom method *compEng()*
            - if it's the *SoftwareEngineeringDept* then in a code block do the following:
                - call *toString()* on the reference prepended with "Dept of: "
                - call the custom method *swEng()*
            - if it's the *SocialCareDept* then in a code block do the following:
                - call *toString()* on the reference prepended with "Dept of: "
                - call the custom method *socialCare()*
            - if it's the *AccountingDept* then in a code block do the following:
                - call *toString()* on the reference prepended with "Dept of: "
                - call the custom method *accounting()*
            - otherwise, throw an *IllegalArgumentException*, outputting the department that is causing the error in the error message.

- *University*
    - in the *main()* method do the following:
        - Force an exception by creating a *LecturerRecord* that has either a blank name and/or a negative age. Test both scenarios. Sample output:

```
Exception in thread "main" java.lang.IllegalArgumentException:
Illegal argument passed:
      "name": ,
      "age": 22

    at assignment.LecturerRecord.<init>(LecturerRecord.java:13)
    at assignment.University.main(University.java:6)
```

```
Exception in thread "main" java.lang.IllegalArgumentException:
Illegal argument passed:
      "name": Joe Bloggs,
      "age": -3

    at assignment.LecturerRecord.<init>(LecturerRecord.java:13)
    at assignment.University.main(University.java:7)
```

- Create a *LecturerRecord* with the following details: the lecturers name is “Jane Bloggs”; she is 24; she works in the Engineering faculty and is in the Software Engineering Department. Output the details by calling *toString()*. Sample output:

```
LecturerRecord[name=Jane Bloggs, age=24, faculty=Engineering, dept=Software Engineering]
```

- Now, rather than using *toString()*, we will output Jane’s details individually by calling each accessor method in turn. Sample output:

```
Name is Jane Bloggs
Age is 24
Faculty is Engineering
Department is Software Engineering
```

- Invoke the *whichFaculty()* on Jane’s reference. Sample output:

```
Faculty of: Engineering
We teach computer science, civil engineering etc...
```

- Invoke the *whichDept()* on Jane’s reference. Sample output:

```
Dept of: Software Engineering
Custom software engineering
```

- Does Jane have a PhD? Sample Output:

```
false
```

- Create a record for “Dr. Anne Bloggs”; she is 35 and works in the Accounting department in the faculty of Business. Output her details (*toString()*) and whether or not she has a PhD. In this case we will decorate the output i.e. rather than simply returning true/false, we will use a ternary operator to output “Ann has a PhD” or “Anne has not a PhD”, depending on true/false respectively. Sample output:

```
LecturerRecord[name=Dr. Anne Bloggs, age=35, faculty=Business, dept=Accounting]
Anne has a PhD
```

- "Joe Bloggs PhD" is 54 and is a member of staff in the Social Care department in the faculty of Humanities. Create a record representing him. Output his details (*toString()*) and whether or not he has a PhD (again, decorate the output). Sample output:

```
LecturerRecord[name=Joe Bloggs PhD, age=54, faculty=Humanities, dept=Social Care]
Joe has a PhD
```