Java Week 3 (2)

Inner classes, Abstract methods and classes, Is-A and class diagrams, interfaces (start)

Plan

- Review Inner classes
- The Is-A relationship
- Abstract classes
- Abstract methods
- Class diagrams
- Interfaces

Inner classes (in more detail)

- AKA Nested Class
- Class written in a class (Outer Class – Inner Class)

```
class Outer {
     class Inner {
     }
}
```

- Static and Non-static (3 versions)

Static Inner Class

- Static class so can be accessed without instantiation

```
class Outer {
     static class Static_Inner {
     }
}
```

- Does not have access to instance variables and methods of the outer class
- Accessed like a method using the . operator

Static Inner Example

Multiple static inners?

```
1  public class Outer_SI {
2      int outerInt = 3; //outer instance variable
3      static class Inner_SI {
4        public void display() {
5            System.out.println("This is a static inner class");
6            //System.out.println(outerInt); //wont compile
7        }
8      }
9
10     public static void main(String args[]) {
11       Outer_SI.Inner_SI innerObj = new Outer_SI.Inner_SI();
12       // use .Inner_SI to instantiate as no inner object
13       innerObj.display();
14     }
15 }
```

```
C:\Users\z2018044\Documents\JavaLa
This is a static inner class
```

Non-static inner classes

- **3 types**
- Inner class
  - private or public
- Method local inner class
  - Neither private or public (method local)
- Anonymous inner class

Inner class (private or public)

- Write the class within a class
- We can make it private (unlike outer classes)
  - Little point if public
- Methods in the outer class can instantiate the inner class
- These outer class methods can be called from the main method in a different class

Inner class (private)

```
1  class Outer_IC { //outer class
2      //static int y = 2;
3      private class Inner_Demo { //private inner class
4          int x = 1;
5          public void print() {
6              System.out.println("I am in inner class " + x);
7          }
8      }
9      void display() { //method accesses a private class
10         Inner_Demo inner = new Inner_Demo();
11         inner.print();
12     }
13
14 }
15
16 public class Test_Outer_IC {
17     public static void main(String args[]) {
18         Outer_IC outer = new Outer_IC();
19         outer.display(); //allows access to methods and variables from private class
20     }
21 }
```

```
C:\Users\z2018044\Documents\Jav
I am in inner class 1
```

Method local inner classes

- A class that only exists inside a method (like a local variable)

```
1  public class Outerclass2 {
2      void oc_Method() { // instance method of the outer class
3          int num = 42;
4          class MethodInner { // method local inner class
5              public void printout() {System.out.println("ic "+num);}
6          }
7          MethodInner inner = new MethodInner();
8          inner.printout();
9      }
10     public static void main(String args[]) {
11         Outerclass2 outer = new Outerclass2();
12         outer.oc_Method();
13     }
14 }
```

```
C:\Users\z2018044\Document
ic 42
```

Method local inner classes

- Why would we do such a thing?

- increase the readability

Anonymous Inner Class

- The (inner) class with no name!
- Declared and instantiated at same time (else how would we identify them?)
- Used for overriding methods…more later

Anonymous Inner Class Method example

```
1  public class AnonymousClassDemo
2  {
3    public static void main(String[] args)
4    {
5      Dog dog = new Dog() {
6        public void someDog () //no semicolon
7        {
8          System.out.println("Anonymous Dog");
9        }
10     }; // anonymous class body closes here
11     //dog contains an object of anonymous subclass of Dog.
12     dog.someDog();
13   }
14 }
15
16 class Dog
17 {
18   public void someDog()
19   {
20     System.out.println("Classic Dog");
21   }
22 }
```

```
C:\Users\z2018044\Documents\JavaL
Anonymous Dog
```

Abstraction

- Dictionary definition of Abstract
  - Thought of apart from concrete realities, specific objects, or actual instances:

    Eg. an abstract idea.
  - Expressing a quality or characteristic apart from any specific object or instance, as justice, poverty, and speed.

- In java we mean the process of hiding the implementation details and only showing what the object does to the user.
- Partly, this is inherent is the message passing nature of java but we also have:
- **Abstract classes and abstract methods**

Abstract Classes

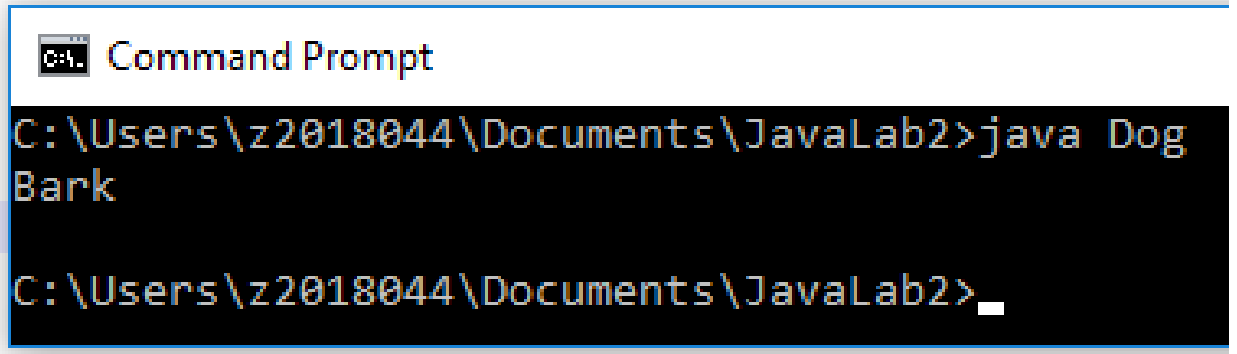
- Abstract classes cannot be instantiated (they produce no objects)
- It needs to be declared `abstract`

```
public abstract class XYZ {…}
```

- Can be subclassed
- These extending classes are called **concrete classes**
- **So an abstract class has no use until unless it is extended by some other class.**
- Useful when every child will override

example

```
2   abstract class AnimalAb{ //abstract
3       public abstract void sound(); //abstract method, doesnt do anything
4   }
5   public class Dog extends AnimalAb{ //inheritance
6
7       public void sound(){
8        System.out.println("Bark");
9       }
10      public static void main(String args[]){
11       AnimalAb obj = new Dog(); //instantiate object
12       obj.sound(); //use the method
13      }
14  }
15
16
```



Abstract Method

- A method with no implementation (it does nothing)
- Only abstract classes can contain abstract methods
- Any subclass of this class must contain implementations for all of the abstract methods in its superclass class..
- ..if not, then it too is abstract (remains abstract?)

- An abstract class can contain abstract and non-abstract methods

Abstract class and methods

```
1  abstract class AnimalAb2{
2      int a = 1;
3      abstract void sound(); //abstract method
4      void travel(){ //concrete method
5        System.out.println("I walk");
6      }
7  }
```

Subclasses of abstract classes

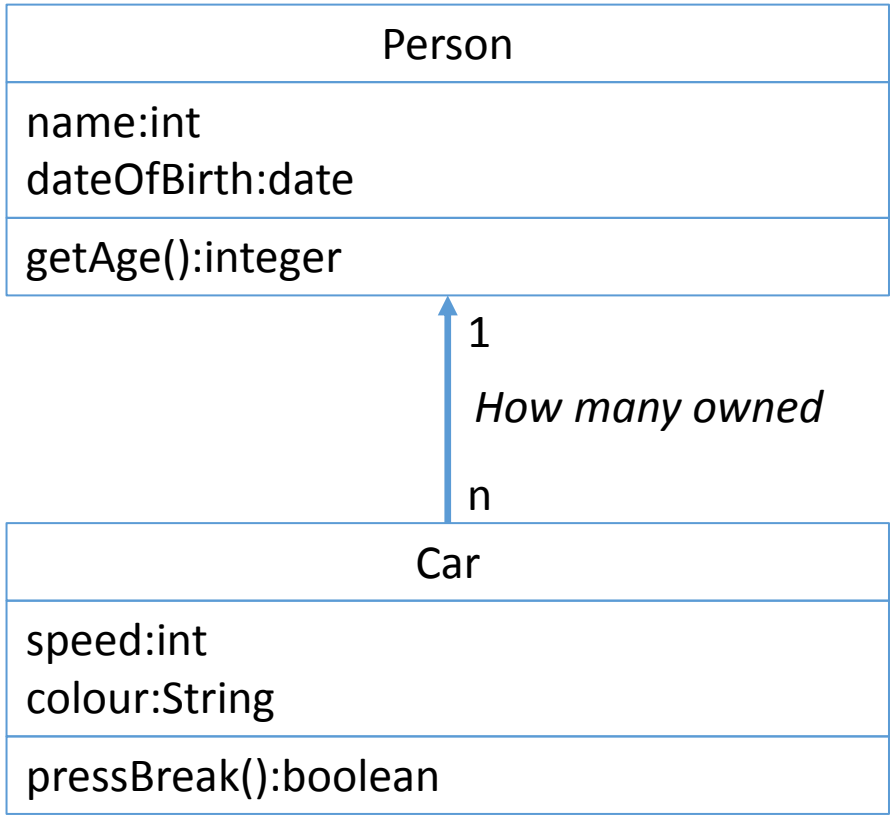
- If the subclass of an abstract class doesn't contain all of its methods then it has to also be an abstract class
- A subclass with all methods from an abstract class no longer needs to be an abstract class

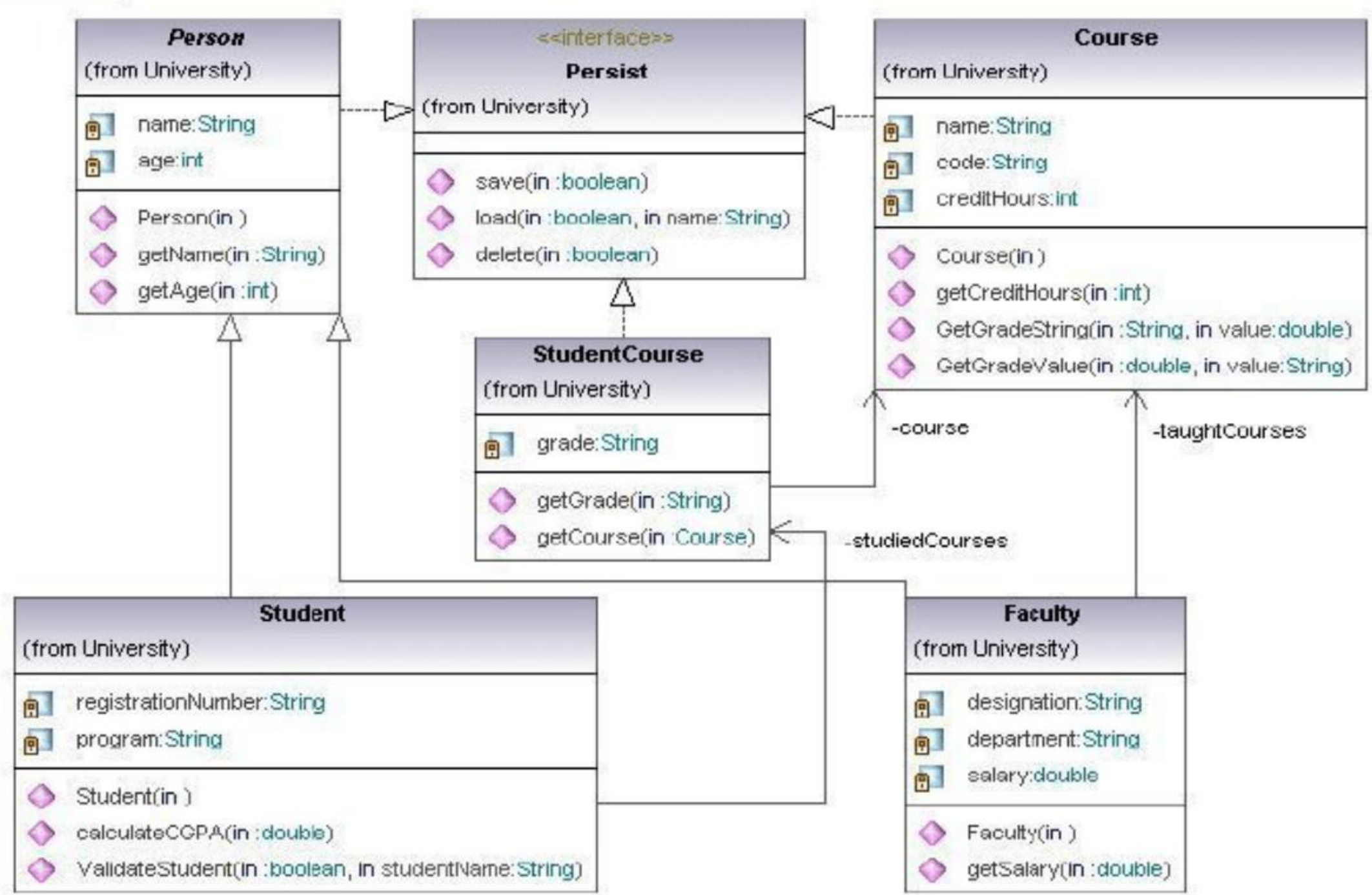
- If you declare an abstract method in a class then you **must** declare the class abstract as well but a class that has no abstract method **can** be marked as abstract.

Class Diagrams

- (part of UML)
- Diagram showing a systems classes
- Each class has 3 compartments
  - The top compartment contains the name of the class.
  - The middle compartment contains the attributes of the class.
  - The bottom compartment contains the operations the class can execute. (methods)

Class diagrams




Person
(from University)
name:String
age:int
Person(in)
getName(in :String)
getAge(in :int)
<<interface>>
Persist
(from University)
save(in :boolean)
load(in :boolean, in name:String)
delete(in :boolean)
Course
(from University)
name:String
code:String
creditHours:int
Course(in)
getCreditHours(in :int)
GetGradeString(in :String, in value:double)
GetGradeValue(in :double, in value:String)
StudentCourse
(from University)
grade:String
getGrade(in :String)
getCourse(in :Course)
-course
-taughtCourses
-studiedCourses
Student
(from University)
registrationNumber:String
program:String
Student(in)
calculateCGPA(in :double)
ValidateStudent(in :boolean, in studentName:String)
Faculty
(from University)
designation:String
department:String
salary:double
Faculty(in)
getSalary(in :double)

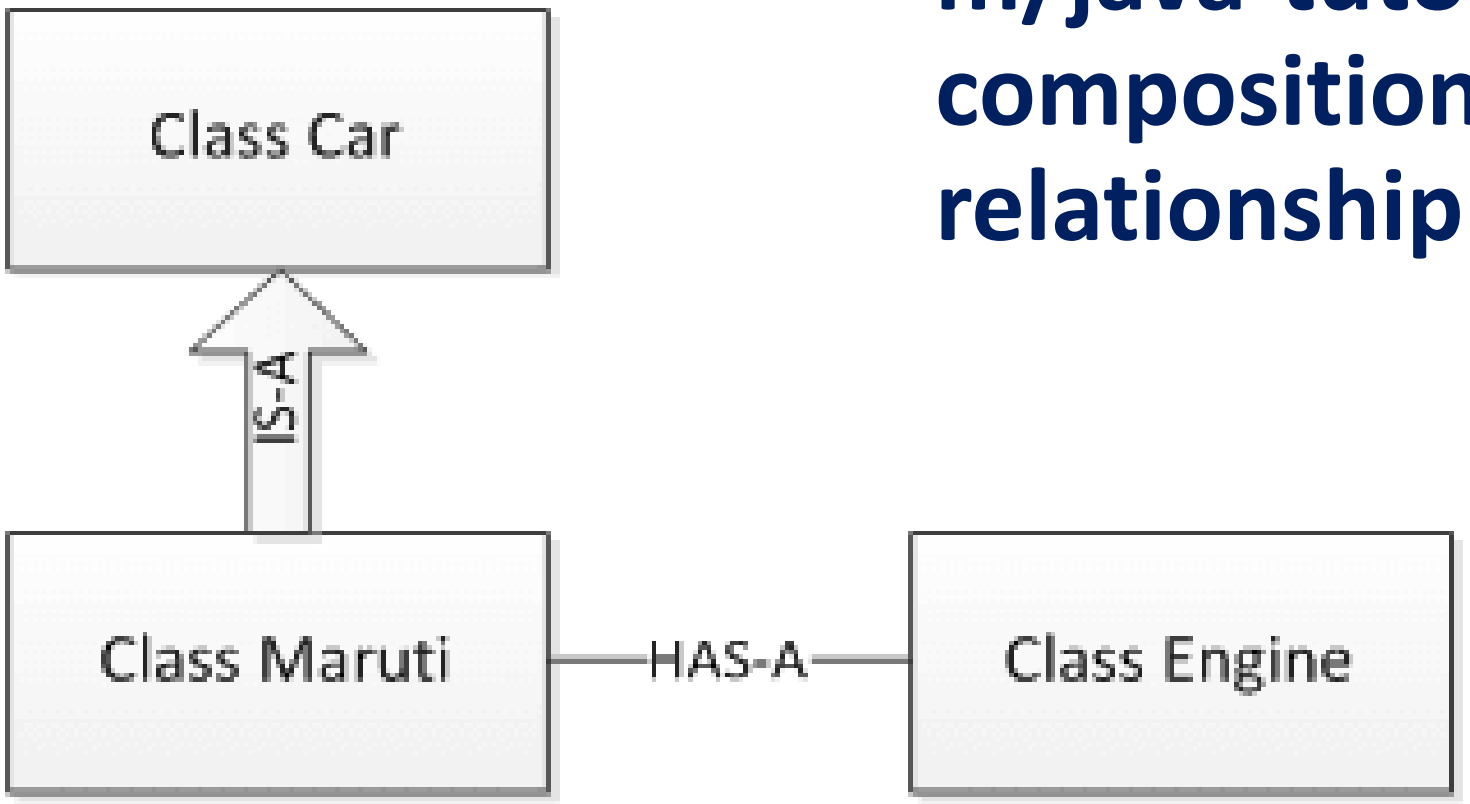
reference

- Usman, M., & Nadeem, A. (2009). Automatic generation of Java code from UML diagrams using UJECTOR. *International Journal of Software Engineering and Its Applications*, *3*(2), 21-37.

The IS-A and HAS-A relationships

- Reusable code, core to OOP
- Whenever you extend class Alpha (or implements) to produce a new class Beta then
  - Beta IS-A Alpha
- This is unidirectional (Alpha IS-A Beta is false)

- Whenever one class uses different public class
  - Beta HAS-A Gamma



https://www.w3resource.com/java-tutorial/inheritance-composition-relationship.php

Interfaces

- It is a collection of methods. (like a Class)
- These methods must be of abstract type
- Abstract methods provide the blueprint but the subclass must provide the implementation
- Interfaces fully separate the interface from its implementation
- Interface defines a set of methods..
- ..but these methods contain NO body

- Main difference between abstract class and interface is that interfaces…..

Useful stuff

- What is OOP – from the guy that first used the term:
  - http://userpage.fu-berlin.de/~ram/pub/pub_jf47ht81Ht/doc_kay_oop_en
  - In two words-”message passing”

- What is message passing?
  - https://softwareengineering.stackexchange.com/questions/140602/what-is-message-passing-in-oo
  - In a few words - objects sending each other messages, the content AND structure of which informs how receiving objects behave