IN1007 Programing in Java

CITY
UNIVERSITY OF LONDON
EST 1894

Lecture 6 (Week 9): Immutable Objects & Intro to Testing

Announcements

- **Second coursework available** → deadline Sunday, 1st Dec at 5pm (see details on Moodle)

- **Final assessment (Viva) →**
  - Special arrangement requests
  - Exact times and room details will be available on Moodle

Module Evaluation Survey

- A formal way to gather your feedback to improve the student experience
- More information on the Student Hub
- Access all open surveys on the Student Survey Portal or via the QR code (you will also be emailed a link)

Today's Lecture

- The `static` keyword (in more detail)
  - Class variables
  - Class methods
  - static initialization blocks
- The `final` keyword
- Immutable Objects
- Intro to Software Testing

Quiz: What does it mean that a field is `static`?

Select **one** correct answer:

- ❑ It cannot be modified once it has been created
- ❑ It is common to a class and not specific to an object
- ❑ It cannot be accessed from outside the class

Click here to answer the quiz

Quiz: What does it mean that a field is `static`?

Select **one** correct answer:

- ☐ It cannot be modified once it has been created
- ☑ **It is common to a class and not specific to an object**
- ☐ It cannot be accessed from outside the class

`Static` variables and methods in Java

- Often referred to as **class-level** variables
- When a **variable** is declared as **`static`**, there is only one copy of it for the entire class (for all its instances).
- When a **method** is declared as **`static`**, it can be called without creating an instance of the class.

- Advantages:
  - Memory efficiency
  - Global access
  - Object independence,…

`static` methods

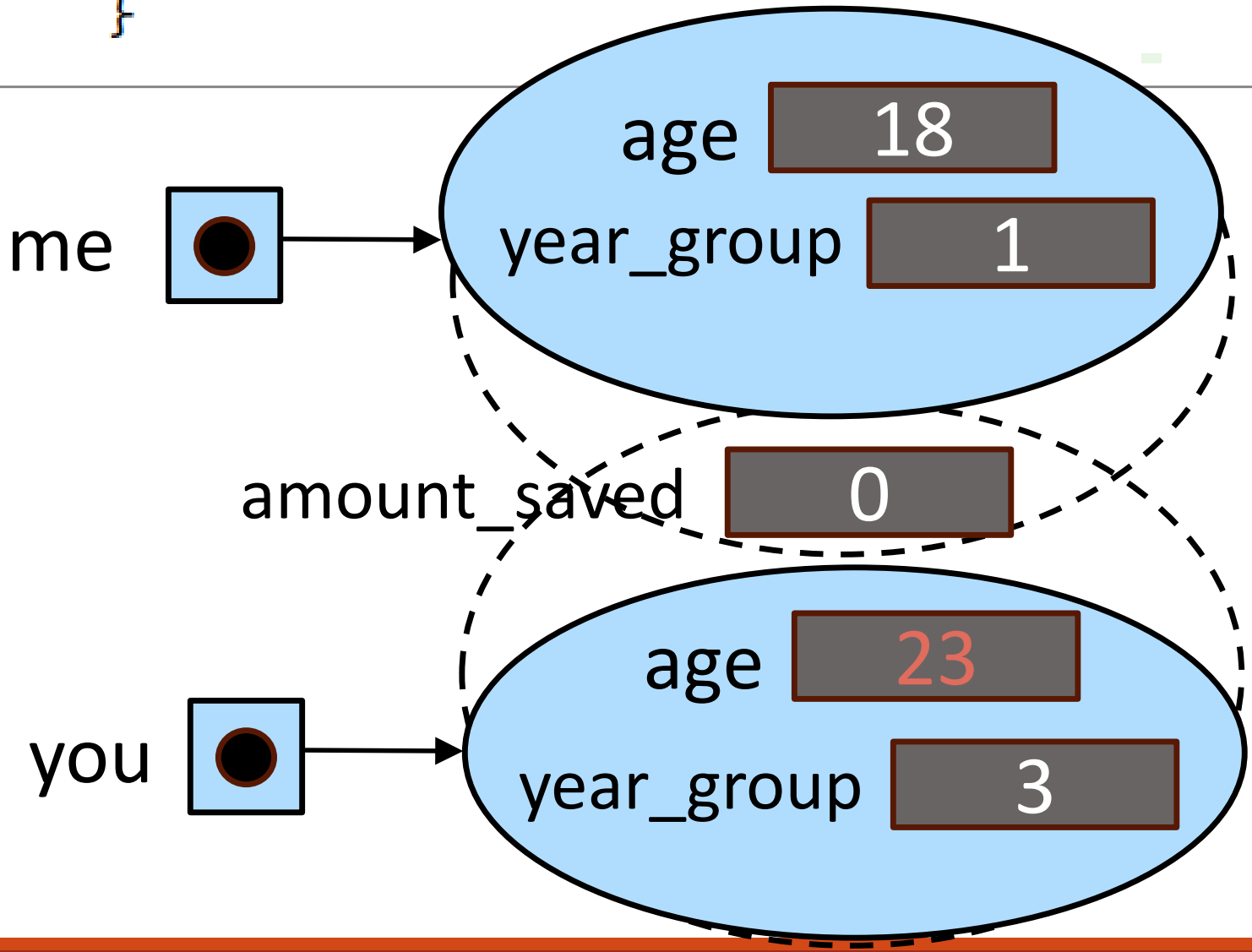
Which of the following is incorrect?

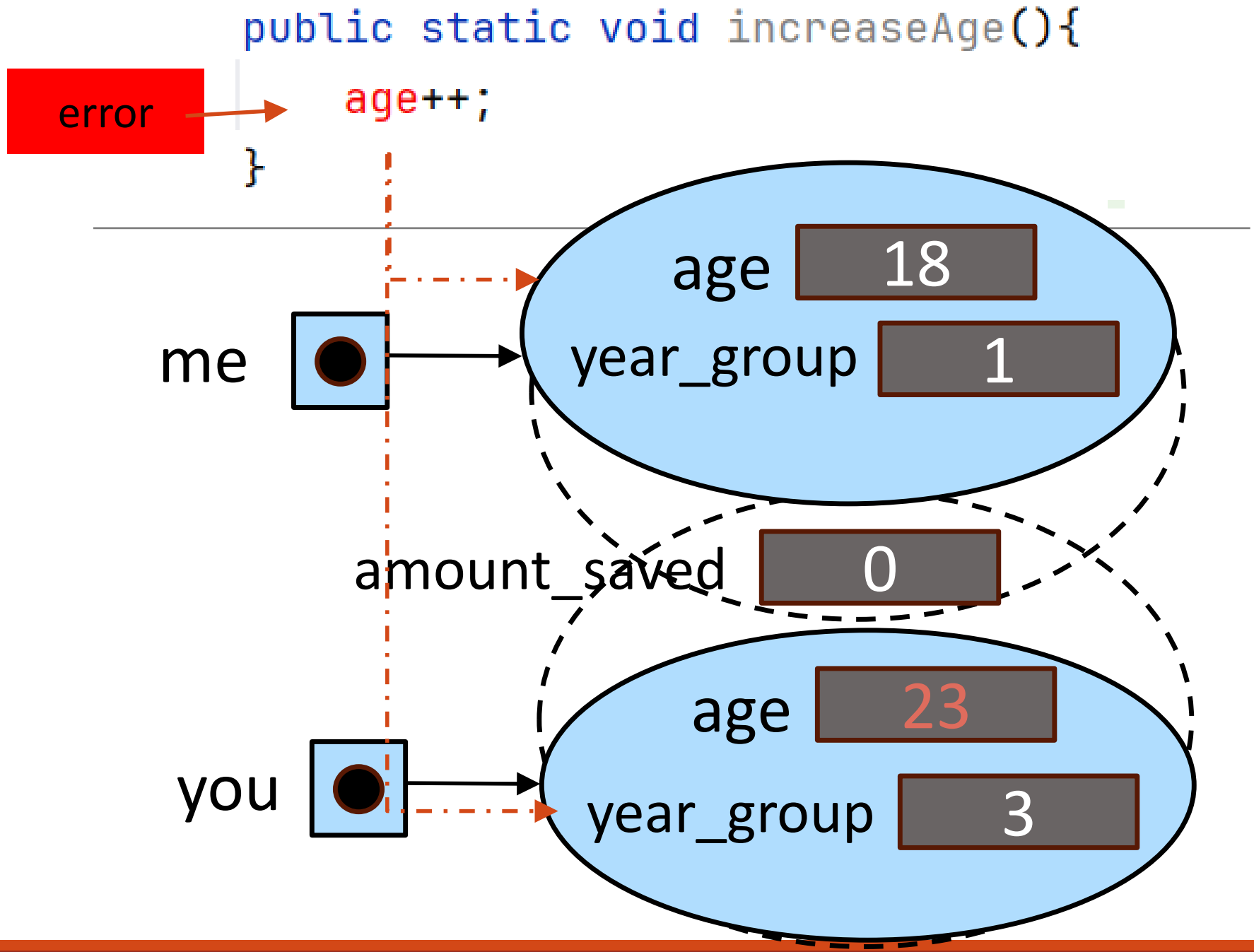
- a static method can access a static field
- a static method can access an instance field
- an instance method can access a static field
- an instance method can access an instance field

Click here to answer this quiz about static methods

```
public static void increaseAge(){
    age++;
}
```

error




public static void increaseAge(){
age++;
}
error
age
18
year_group
1
me
amount_saved
0
age
23
you
year_group
3

Referencing variables

- Instance variables / fields are referenced by

`ObjectName.variableName`

Example: `me.year_group`

- Static variables / class variables are referenced by

`ClassName.variableName`

Example: `Student.amount_saved`

Rules for instance members & methods

- A non-static member of a class is called an instance member
- Instance methods can access instance variables and instance methods directly
- Instance methods can access static variables directly
- Instance methods can call static methods directly

Rules for `static` members & methods

- `static` methods can access `static` variables directly
- `static` methods can call `static` methods directly
- `static` methods **CANNOT** access instance variables directly
- `static` methods **CANNOT** call instance methods directly. They must use an object reference.
- `static` methods **CANNOT** use the **`this`** keywork as there is no instance to refer to.

Try adding this method to the Student class from last lecture (also on Moodle)

```
public static void increaseAmount(){
    amount_saved++;

}
```

See what breaks:

- Try accessing an instance field like `age`
- Try calling and instance method like `getAge()`
- Try calling this method from an instance method

Today's Lecture

- The `static` keyword (in more detail)
  - Class variables
  - Class methods
  - **static initialization blocks**
- The `final` keyword
- Immutable Objects
- Intro to Software Testing

`static` initialization blocks

```
static {
  ...
}
```

→ Run once, **when the class is loaded for the first time**, independent of how many instances of the class are created.

Initializer Blocks in Action

Try adding the following non-static initializer block in the class Student:

```
{
    System.out.println("Hello from a student");
}
```

Try this:

- Create two instances of Student in `main()`
- Create one instance of Student in `main()`
- Create zero instances of Student in `main()`

Is this a static block?

Initializer Blocks in Action

Try adding the following `static` initializer block in the class Student after the previous block:

```
static {
    System.out.println("Hello from a static block");
}
```

Observe:

- Which block gets executed first?
- How many times does the static block get executed?
- How many times does the instance block get executed?

Today's Lecture

- The `static` keyword (in more detail)
  - Class variables
  - Class methods
  - Static initialization blocks
- **The `final` keyword**
- Immutable Objects
- Intro to Software Testing

Constants in Java

- In Java, all variables of primitive data types (e.g., int, float, double, boolean, etc.) are modifiable.
- Example:

```
int x = 5;
x = 4;
x = x -1;
```

- Sometimes you may wish to declare a **constant** rather than a variable.
- A constant has a set value that does not change
- Example: if you want to define the constant Π = 3.14159265

Constants in Java

Use the `final` keyword to define a constant which cannot have their value changed

This is commonly used along with the `static` keyword. **Can you think of why?**

Naming convention: constants are usually defined using UPPERCASE letters.

```
static final double PI = 3.14;
```

Today's Lecture

- The `static` keyword (in more detail)
  - Class variables
  - Class methods
  - static initialization blocks
- The `final` keyword
- **Immutable Objects**
- Intro to Software Testing

Break

Today's Lecture

- The `static` keyword (in more detail)
  - Class variables
  - Class methods
  - static initialization blocks
- The `final` keyword
- **Immutable Objects**
- Intro to Software Testing

What is an immutable object?

Definition:

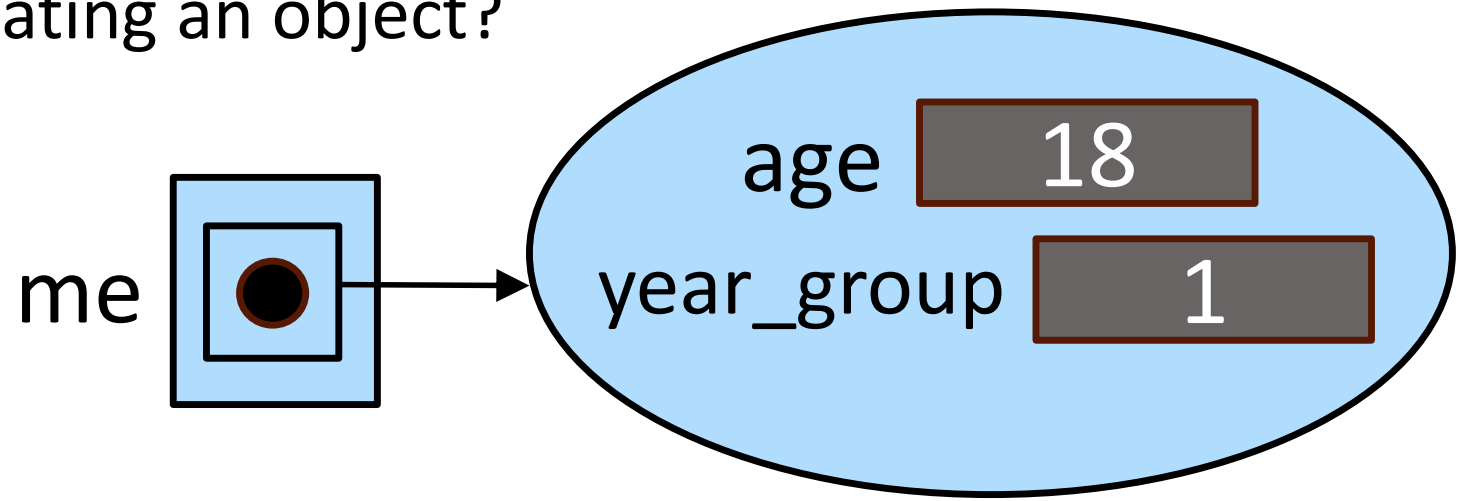
Immutable objects are objects which cannot be changed once they have been created.

Question:

How do we implement an immutable object in Java?

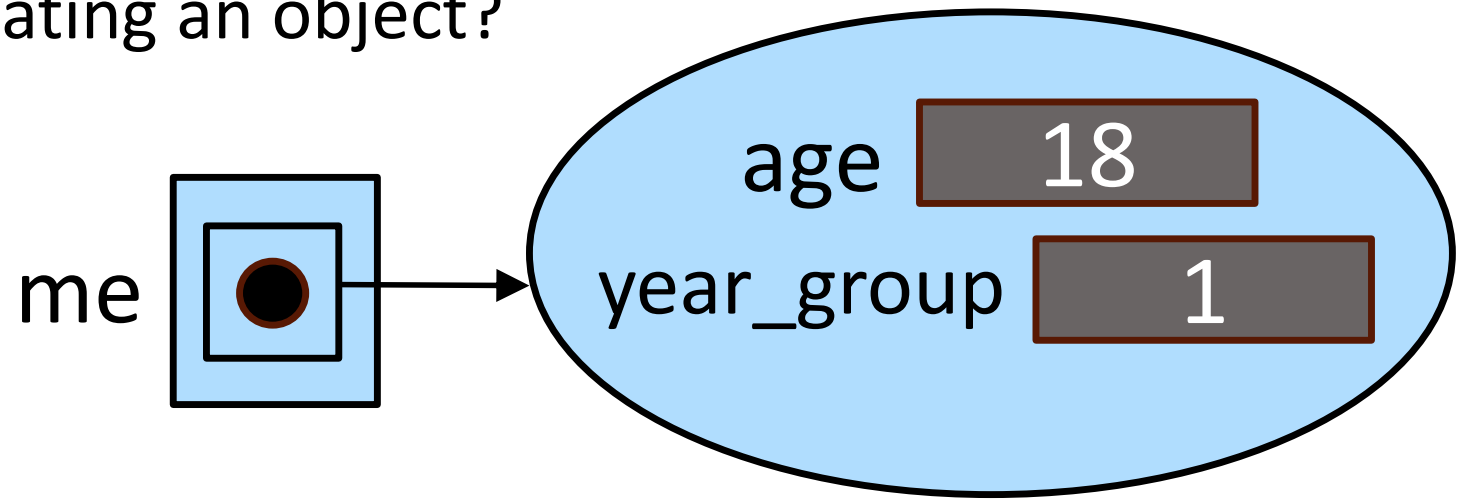
Observe…

What happens if we use the keyword `final` when creating an object?



Observe…

What happens if we use the keyword `final` when creating an object?



It simply means the reference of the object cannot change BUT you can still modify the state of the object.

How?

Testing a `final` object

```
public class Main {
    public static void main(String[] args){
        Student you = new Student("John", 23,3);
        final Student me = new Student();
        me = you;
    }
}
```

What error does this produce?

Testing a `final` object

```
public class Main {
    public static void main(String[] args){
        Student you = new Student("John", 23,3);
        final Student me = new Student();
        me.setAge(15);
    }
}
```

Does this give an error?
Does it fulfill the requirement for immutable objects?

So, how do we implement an immutable object?

1. Ensure encapsulation is upheld so an object's state is only accessible via its methods → but that's not enough

So, how do we implement an immutable object?

1. Ensure encapsulation is upheld so an object's state is only accessible via its methods → but that's not enough

2. Remove the setters (methods that can modify the state)

So, how do we implement an immutable object?

1. Ensure encapsulation is upheld so an object's state is only accessible via its methods → but that's not enough

2. Remove the setters (methods that can modify the state)

3. Use `final`

So, how do we implement an immutable object?

1. Ensure encapsulation is upheld so an object's state is only accessible via its methods → but that's not enough

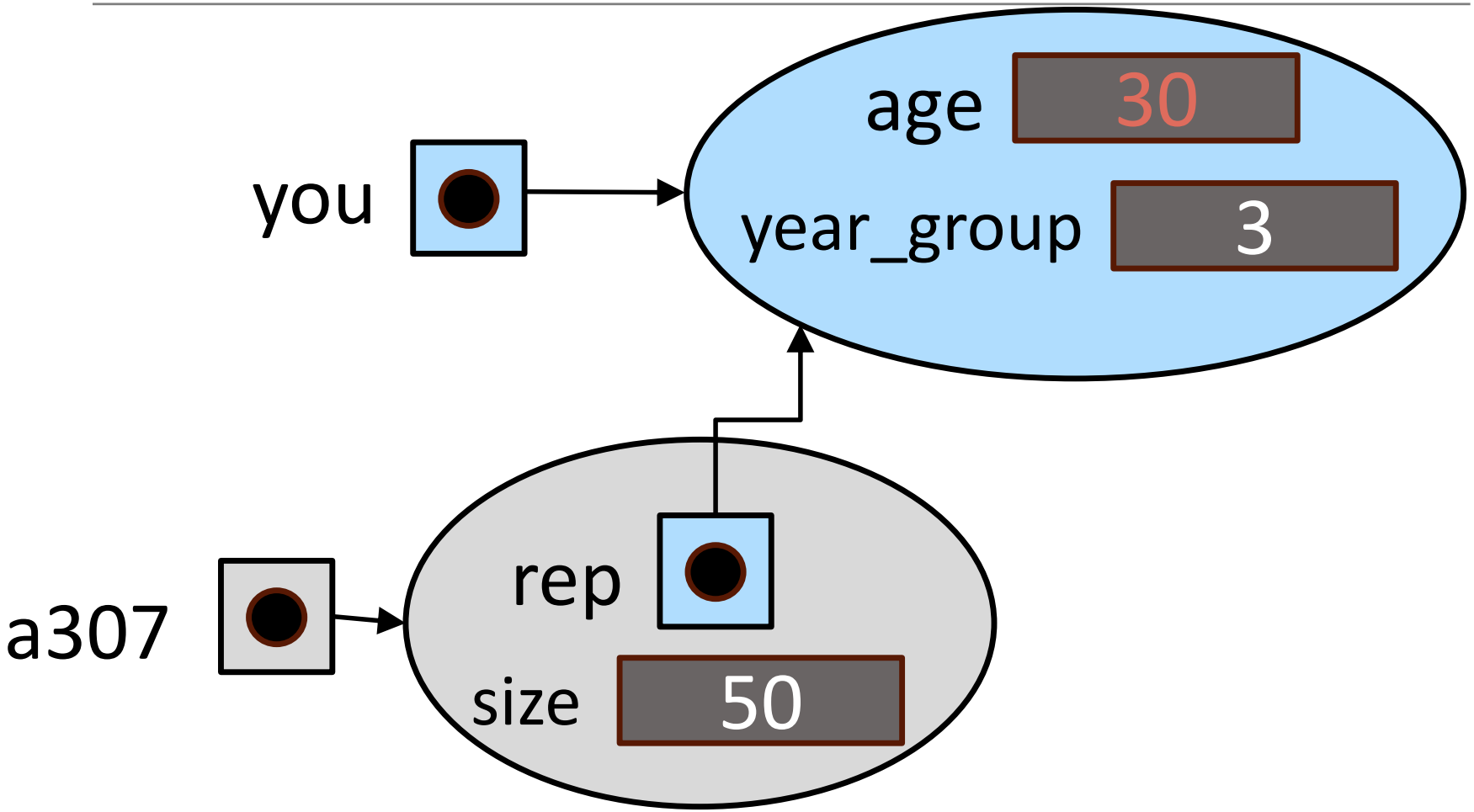
2. Remove the setters (methods that can modify the state)

3. Use `final`

4. For member fields that are object references, do not allow these objects to be modified

Let’s unpack number 4….

Suppose we want to make the a307 object immutable



Start with the easy bit…

1. Ensure encapsulation is upheld so an object's state is only accessible via its methods → but that's not enough

2. Remove the setters (methods that can modify the state)

3. Use `final`

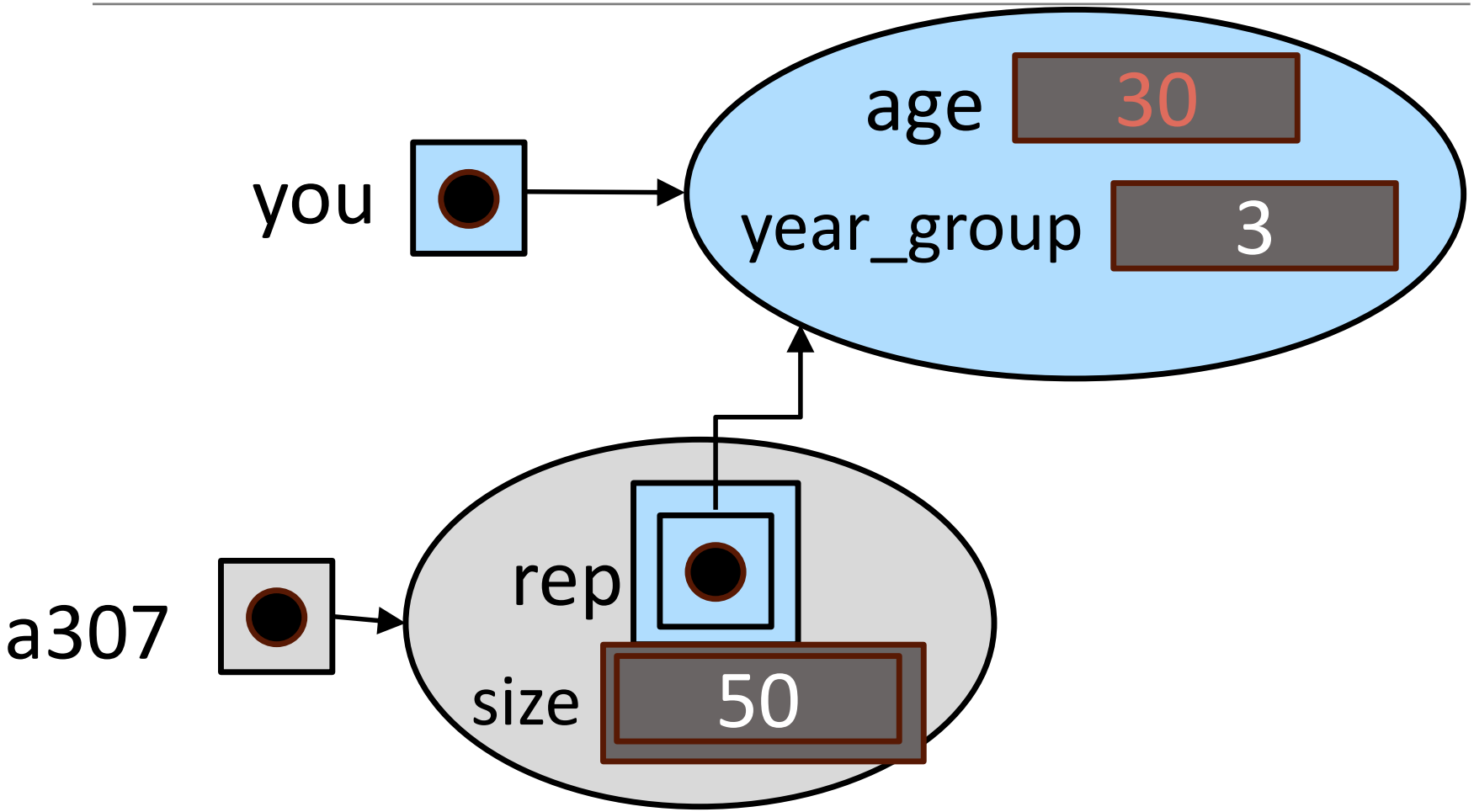
Removed `setRepAge()`

```
public class Classroom {
    private final int size;
    private final Student rep;

    Classroom(int size, Student rep){
        this.size = size;
        this.rep = rep;
    }
    public int getRepAge() { return rep.getAge(); }
}
```

But remember…

There is still another reference available (`you`)



What can we do?

Option 1:

Make the Student object immutable

Option 2:

NEVER allow the reference to the mutable object to be stored somewhere

Option 1: Do it yourself

Try modifying the Student class to make it immutable

Remember the strategy:

1. Ensure encapsulation is upheld so an object's state is only accessible via its methods → but that's not enough

2. Remove the setters (methods that can modify the state)

3. Use `final`

4. For member fields that are object references, do not allow these objects to be modified

Option 2: Without modifying Student.java make the following changes in Classroom.java

```
Classroom(int size, Student rep){
    this.size = size;
    this.rep = new Student();
    this.rep.setAge(rep.getAge()); //etc.
}
public Student getRep(){
    Student newrep = new Student();
    newrep.setAge(this.rep.getAge()); // etc.
    return newrep;
}
```

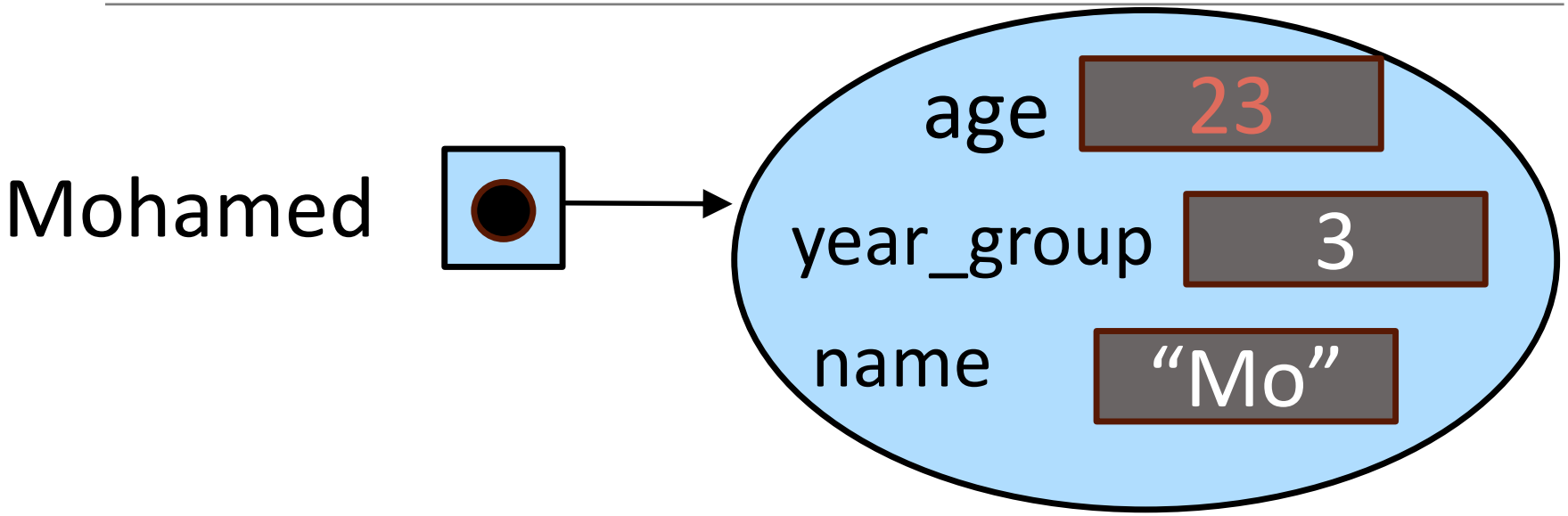
Discuss

Work with one of your classmates to sketch the copies of a `Student` object that get created in memory as:

(i) a Classroom object gets created and

(ii) the `getRep()` method gets called in `main()`

```
Classroom(int size, Student rep){
    this.size = size;
    this.rep = new Student();
    this.rep.setAge(rep.getAge()); //etc.
}
public Student getRep(){
    Student newrep = new Student();
    newrep.setAge(this.rep.getAge()); // etc.
    return newrep;
}
```


Mohamed
age
23
year_group
3
name
“Mo”

Exercise from last lecture

1. Implement a public class `Restaurant` with the following private fields:
   - `name` of type `String`,
   - `numberOfTables` of type `int`.
2. Provide a constructor with one parameter of type String and that initializes the field `numberOfTables` to 10.
3. Provide getter and setter for the fields.
4. Modify the setter method for `numberOfTables` so that this field is updated only if the argument is a positive integer.
5. Add an initialization block that prints "Welcome to the new restaurant".
6. Implement another public class `Test` with a main method in which you create a new object of the class `Restaurant` and test your setter and getter methods.

Try this!

1. In your class `Restaurant`, add a static private field `numberOfRestaurants` and getter and setter for this field.
2. Implement a static private method to increment the field `numberOfRestaurants`.
3. Call this method in every constructor.
4. Test your code.
5. Implement a static initialisation block which prints "Welcome".
6. In your main method, test it.

Today's Lecture

- The `static` keyword (in more detail)
  - Class variables
  - Class methods
  - static initialization blocks
- The `final` keyword
- Immutable Objects
- **Intro to Software Testing**

What is Software Testing?

"Software testing is an empirical technical investigation conducted to provide stakeholders with information about the quality of the product or service under test"

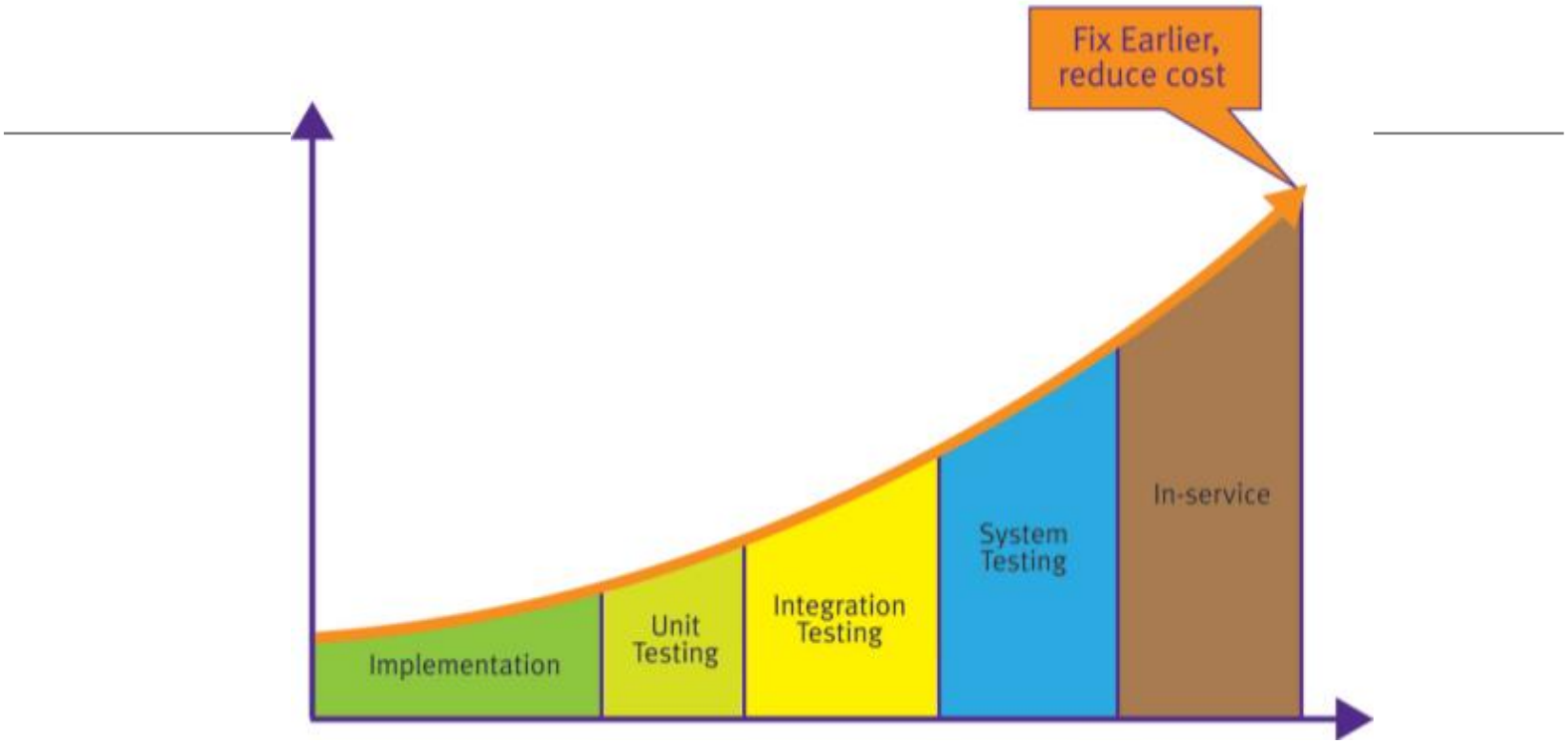
Professor Cem Kaner - Director of Florida Tech's Center for Software Testing Education & Research

- **Empirical** - derived from experiment, experience, and observation
- **Technical** - Requiring special practical knowledge
- **Investigation** - detailed inquiry or systematic examination

Why testing is Important

- All Software products have defects (bugs)
- Software products are getting larger and more complicated
- Software Engineering is not as mature as other disciplines e.g. Civil Engineering
- Software is written by people – people make mistakes
- Software testing looks to find the most important defects as early as possible – increasing confidence that the software meets specification

The cost of defects



The later a defect is found in the product development, the more costly it is to fix . This is a concept first established in 1975 with the publication of Frederick Brooks' "Mythical Man Month" and proven many times since through various studies.

(Image source: embeddedinsights)

Fundamentals of Software Testing

- Software testing needs planning, tests need specifying, once executed they need results recording, and post completion should be easily auditable

Different Levels (/Types) of Testing – When do you test?

- **Unit Testing** – At the time of writing code
- **Integration Testing** – When different pieces of code need combining
- **Validation Testing** – while working on a release/version – throughout development
- **Acceptance Testing** – generally at/before a major version/release
  - Alpha Testing
  - Beta Testing

Note: Not an exhaustive list.

Unit Testing

- Algorithms and logic
- Data structures (global and local)
- Interfaces
- Boundary conditions
- Error handling

Unit Testing

- Isolate & test individual units of code
- A unit is the smallest testable part of a program
- In OOP, individual methods would be the units
- Can begin testing before entire program is done

Extra Curricular Activity….

Design a Unit testing strategy to test the immutability of the Student class

You may refer to:

Testing Object's Immutability: Is This a Good Idea? – The Coders Tower