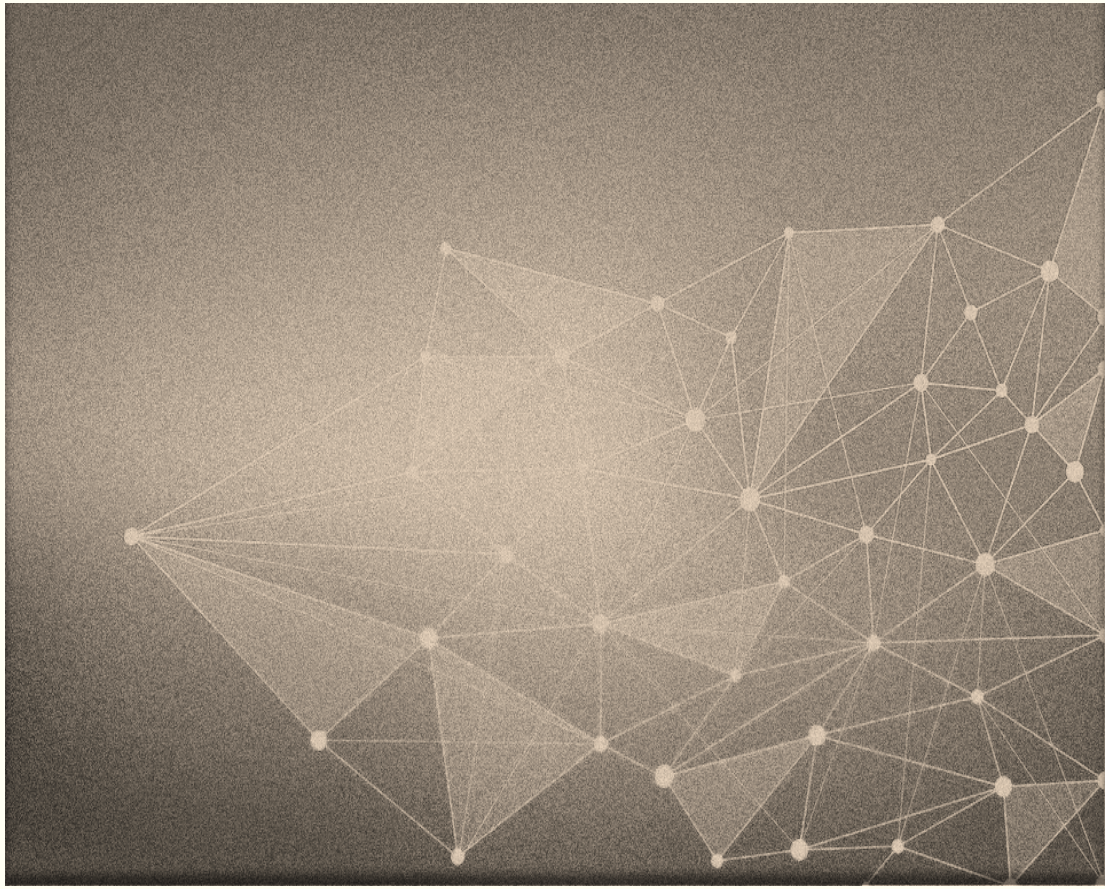
EECS2030:
ADVANCED OBJECT-ORIENTED PROGRAMMING

By: Dr. Marzieh Ahmadzadeh

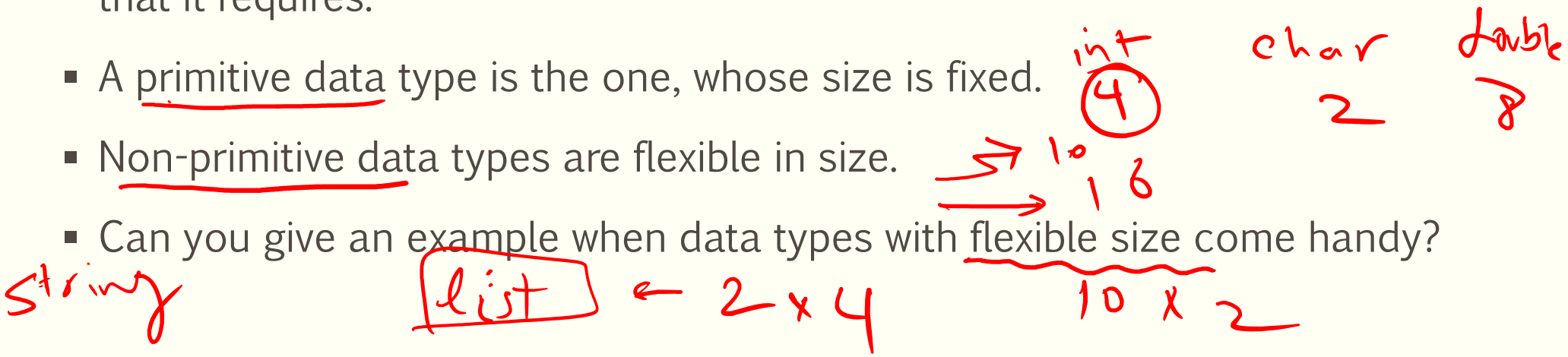


Outline

- Last Lecture
  - Recursion
    - How to design
    - How to trace
    - How the memory looks like when a recursive function is called.
- This week's outline
  - Memory diagram for primitive and non-primitive
  - Classes & Objects
  - Constructors
    - Default, copy constructor, constructor chaining and overloading constructors
  - Encapsulation
    - Accessor and mutator methods
  - In-Class Activity

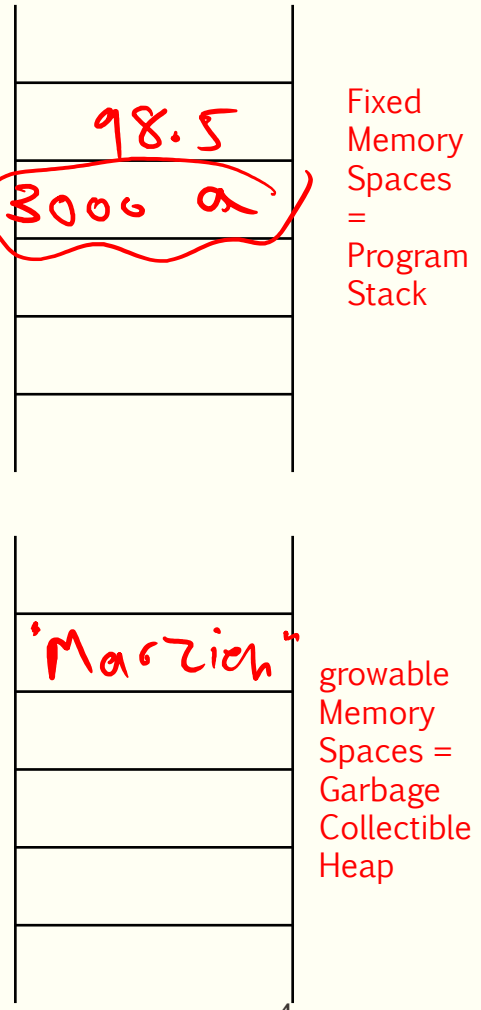
Variables Types

- Java cares about the type. Java is a strongly typed language.
- Variables must have a type to indicate the size of the container (i.e. memory space) that it requires.
- A primitive data type is the one, whose size is fixed.
- Non-primitive data types are flexible in size.
- Can you give an example when data types with flexible size come handy?



Non-primitives

- There are a lot of non-primitive built-in data types that you are familiar with
  - String, Math, Scanner, ArrayList, Vector, StringBuilder, StringBuffer
- Non-primitive data types are also called `reference types`.
  - Why do you think is the rational behind this naming?

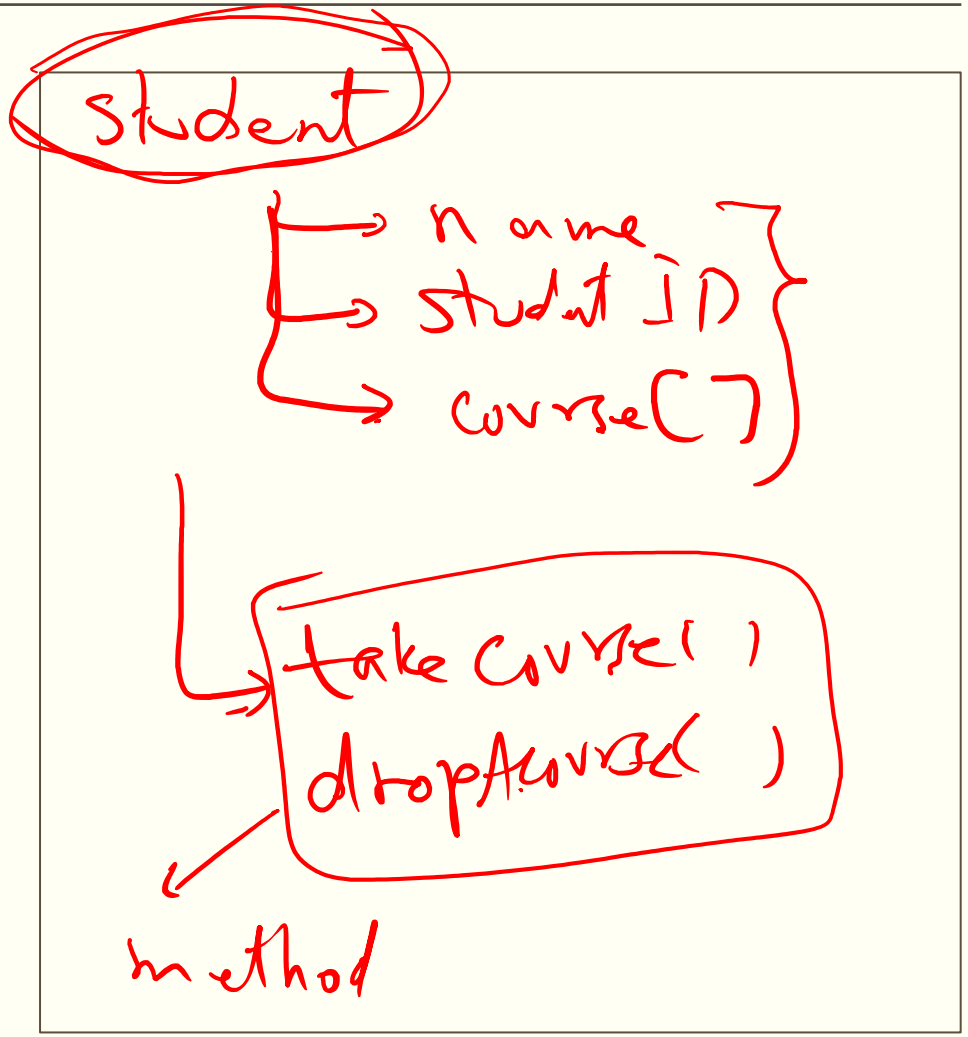


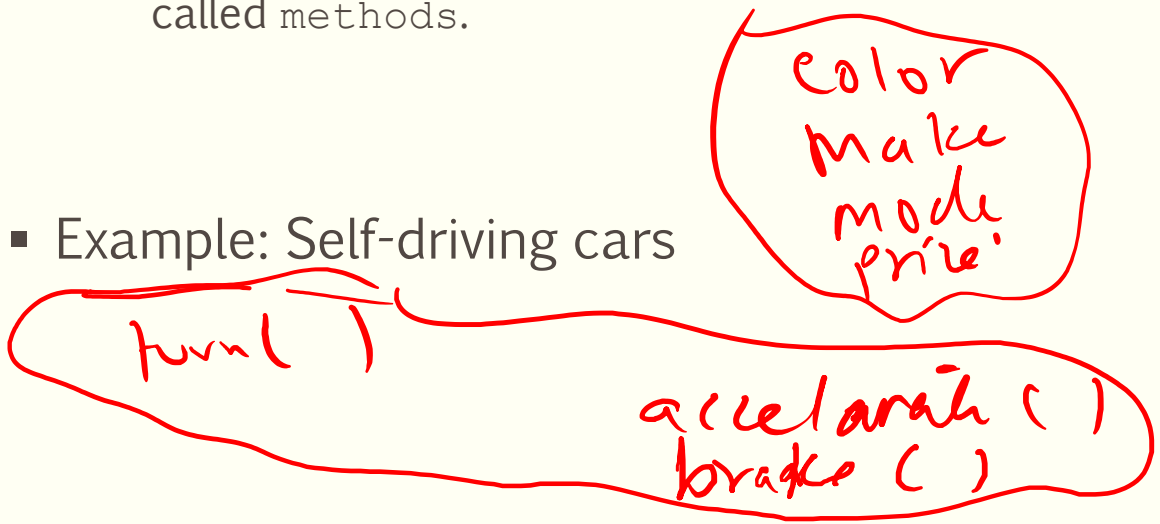

```
public static void main (String[] args) {
    double grade = 98.5;
    String name = "Marzieh";
}
```

Question: What will be the address of “name” in the stack?

Non-primitive data types

- Reference types have zero or more states.
  - Their states are presented by their `instance variables` (AKA `fields`, `attributes`)
- There are zero or more behaviours associated with them.
  - Behaviours are presented by functions.
  - In object-oriented programming, functions are called `methods`.
- Example: Self-driving cars





Classes = non-Primitive Datatype

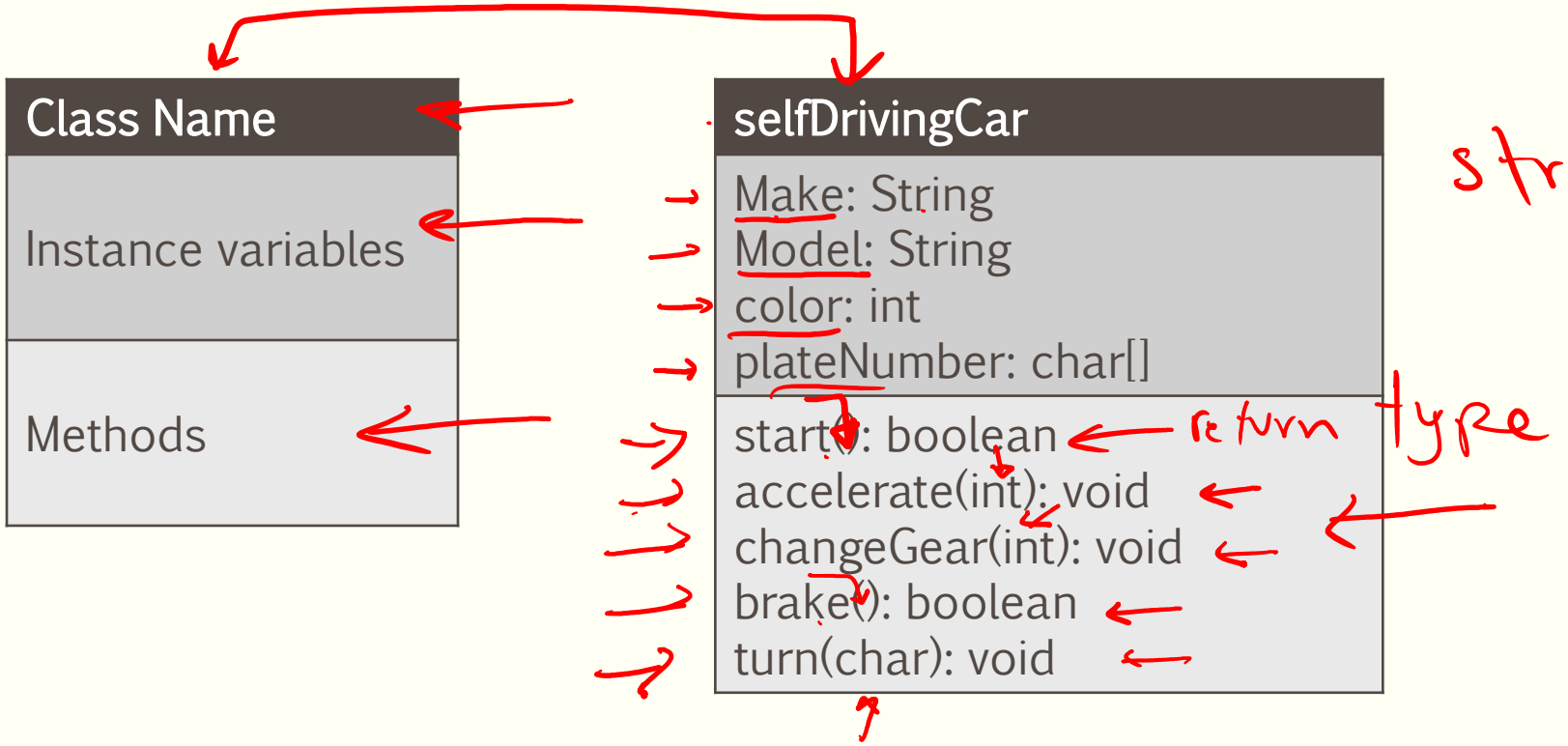
- A bundle of instance variables and methods is called a `class`.
- A `class` is a data type.

- Example: Person, Student, Car, String, Math

```
class Person{
    String name;
    int age;
    void walk(){ }
    void talk(){ {
{

Person john;
Person jane;
```

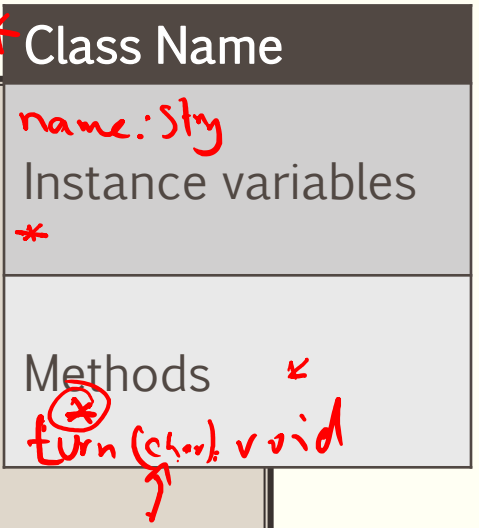
Presenting a Class

- The notation that is used to present a class is called UML.



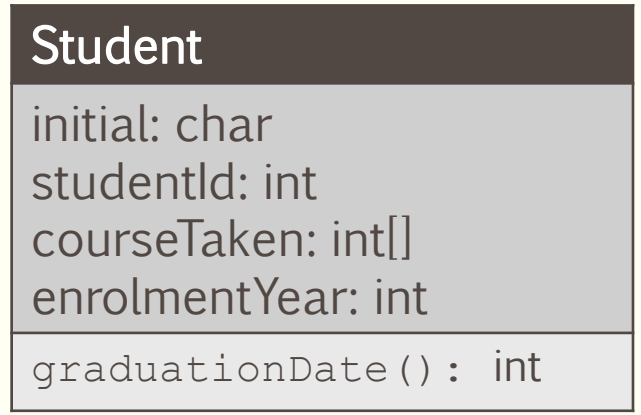
Creation of a class

```
[public] class className{

    type variable_name;

    [access modifier][static] returnType/void methodsName(type name, …){
    } // end of method


}// end of class
```



Creation of a class

```
/**
 * This class introduces a student
 *
 */
class student {
    char initial;
    int studentId;
    int[] courseTaken;
    int enrolmentYear;

   /**
    *
    * @return returns the year of graduation for this student
    */

    public int graduationDate() {
        return enrolmentYear + 4;
    }
}
```



Objects

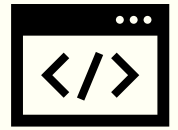
string name;

- A variable defined using a class (non-primitive/ reference type) is called an `object`.
  - An object is an instance of a class.

student John;

- To make an object '`new`' operator is used followed by the name of the class.

- To get access to the objects' property (i.e. fields & methods), `dot` '.' operator is used.
- With one class (reference type), you can define as many objects as you require.



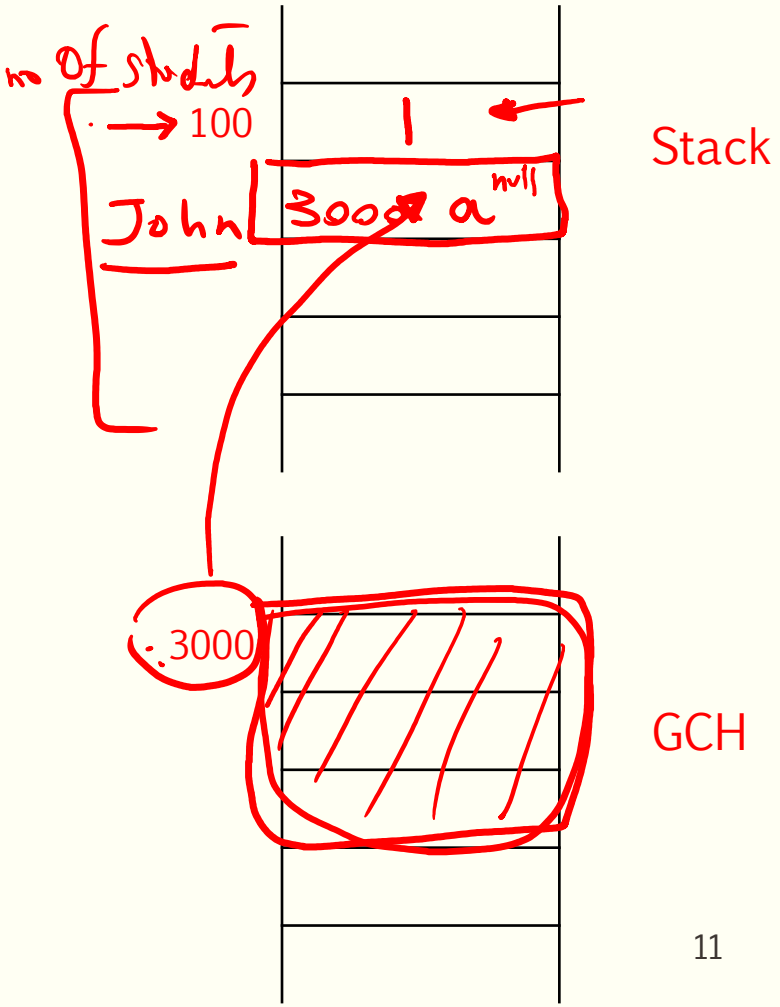
Memory Diagram/Model

- Just like primitive variables, when you define an object you tell java to reserve a space in memory for the object.
  - The keyword to order the reservation is 'new'.
  - In which part of the memory, the space is reserved?

```
public static void main(String[] arg) {
    int noOfStudents = 1;
    Student john = new Student();
}
```

- Creating an object is a 3 steps process:
  - Declaration: `Student john`
  - Space reservation: `new Student();`
  - Address Assignment: `Student john = new Student();`

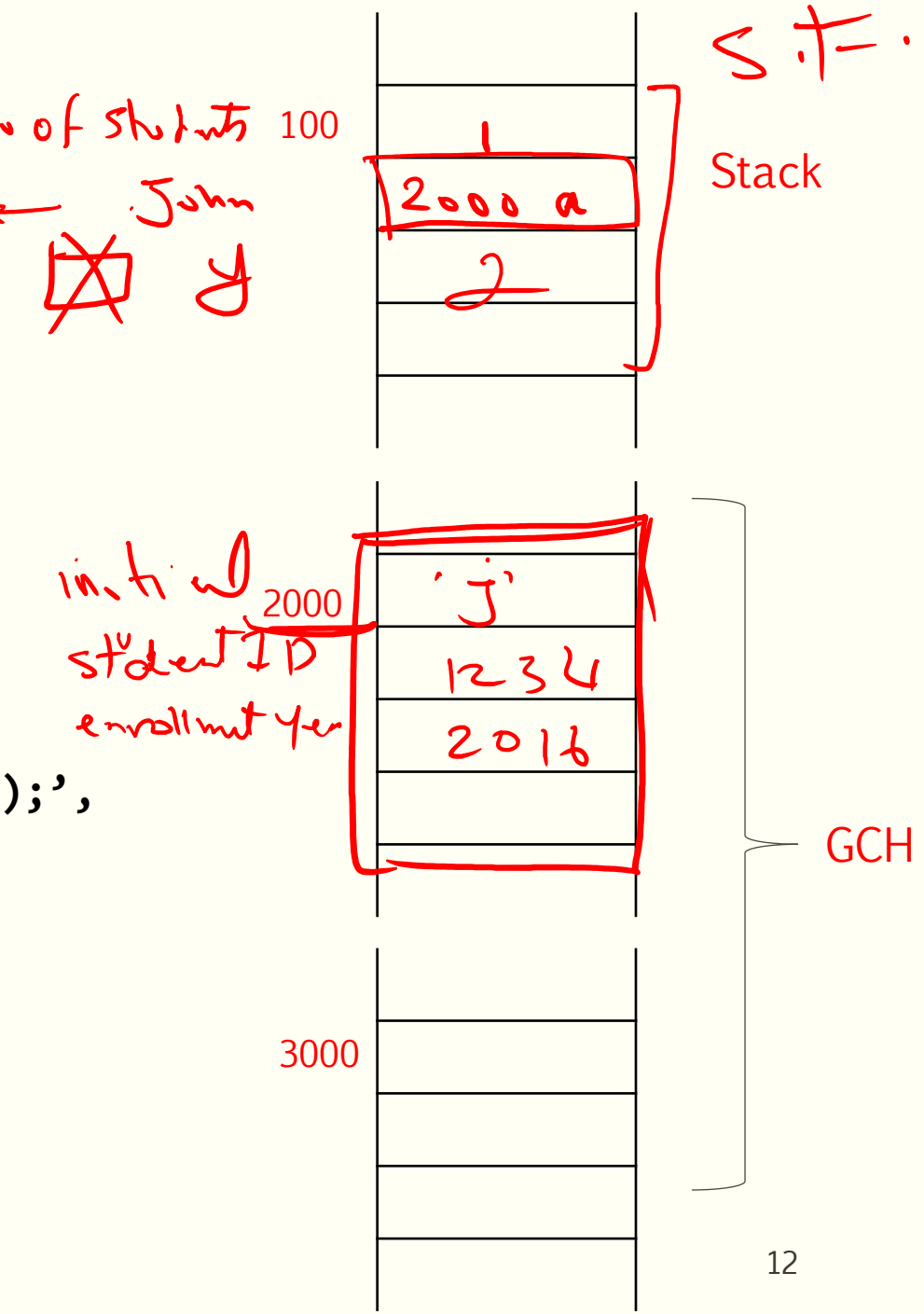
Local variable Of functions → Stack



Memory Diagram/Model

```
int noOfStudents = 1;
Student john = new Student();
john.initial = 'J';
john.studentId = 1234;
john.enrolmentYear = 2016;
```

If I add 'int y = 2;' after 'Student john = new Student();', what would be the address for y?

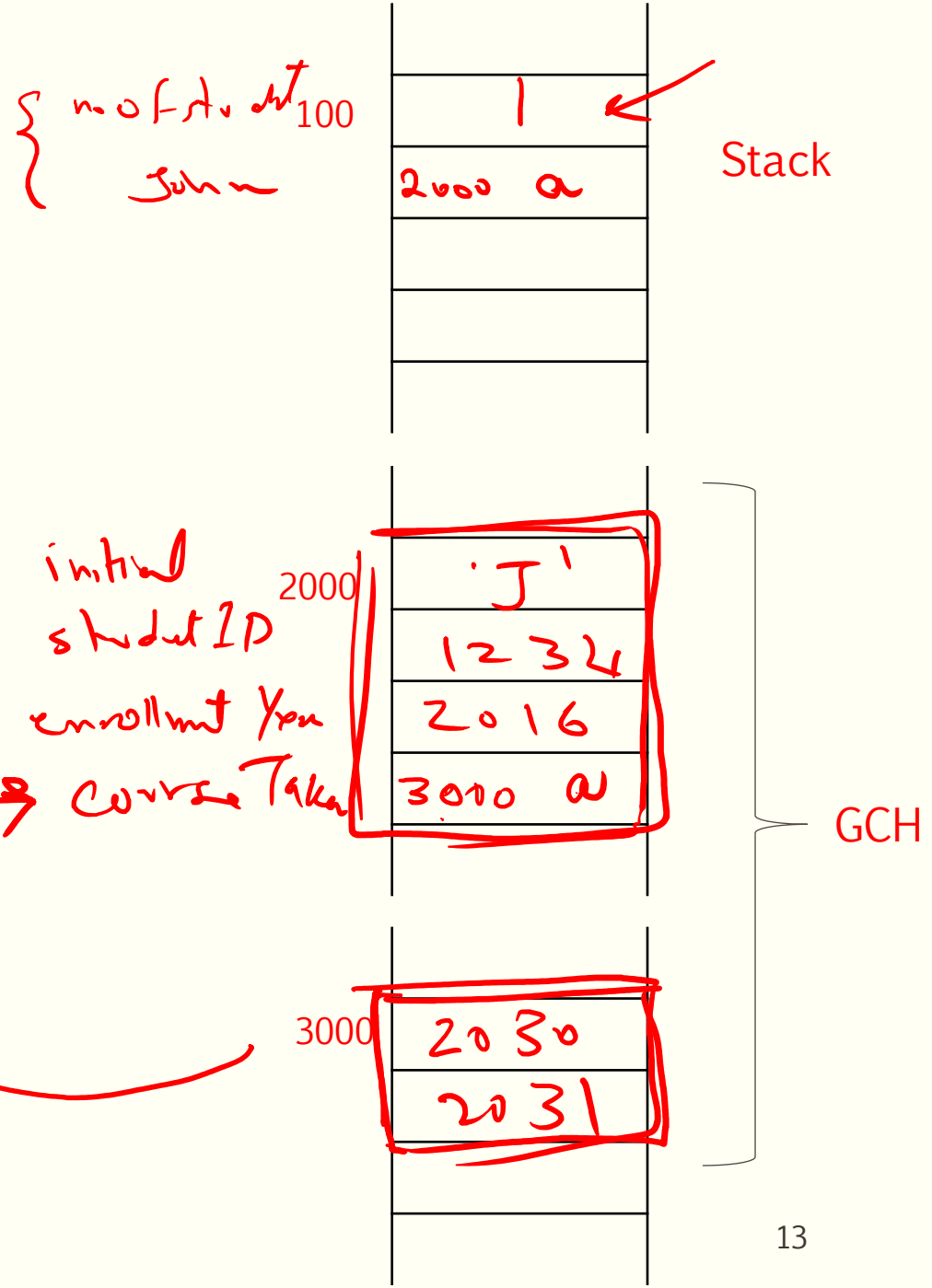


Arrays are non-primitive data type

- Just like other reference types, arrays need to request for a memory space.

- Example: initialize the `courseTaken` array in class student.

```
int hoOfStudents = 1;
Student john = new Student();
john.initial = 'J';
john.studentId = 1234;
john.enrolmentYear = 2016;
john.courseTaken = new int [2];
john.courseTaken[0] = 2030;
john.courseTaken[1] = 2031;
```



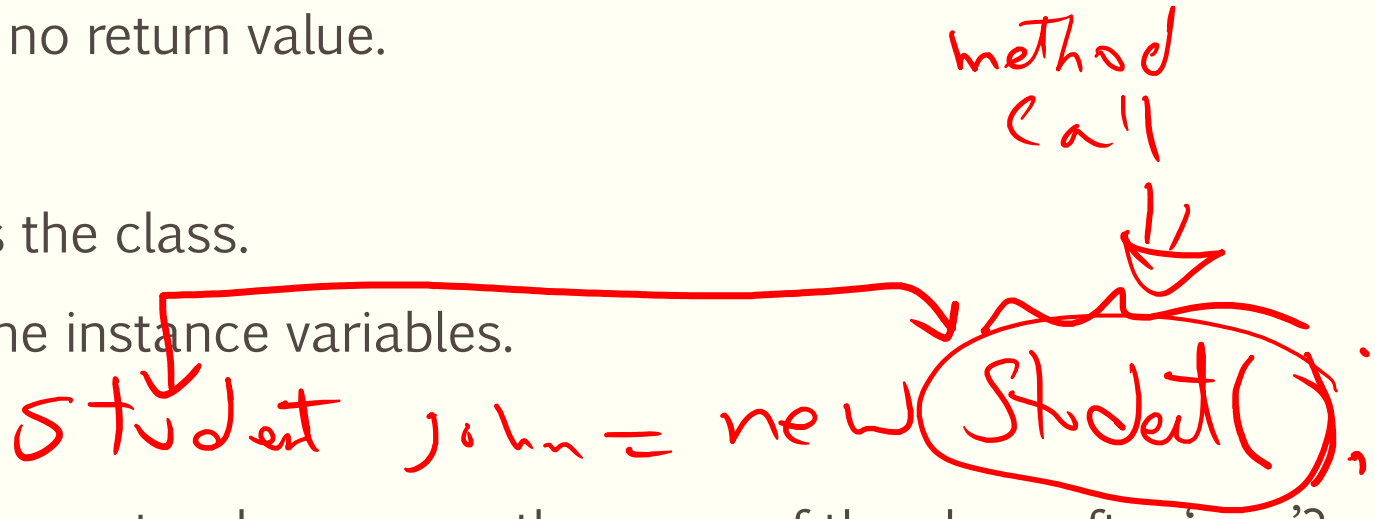
In-Class Activity 1

Summary

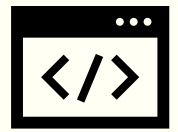
- A class is a blueprints for objects.
- A class represent a data type by which you can define a variable.
- The variable that you define using a class, is called an object.
  - This variable holds a reference to the actual object.
- You define one class (i.e. Student) by which you can define as many objects as you need (i.e. all the students who enrolled in EECS2030)
- Up to now, you used built-in classes (e.g. String, Math, Scanner, etc) to define an object, but from now on you can create your own classes.
- It's good to know: For simplicity in the memory diagram presented in this course both GCH and Stack grow down when a variable is added. In reality, however, stack grows down, and heap grows up.

Constructors (1)

- A special method with no return value.
- It is not a void.
- It is the same name as the class.
- It is used to initialize the instance variables.



- Does it make sense now as to why you use the name of the class after 'new'?
  - new Student();
- When I didn't have any constructor for Student, I didn't get any error for `new Student();` why?
- This type of the constructor, in which you initialize the attributes to the default values, is called `default constructor`.



Constructors (2)

To combine the object creation and instance variable initialization, a constructor can accept parameters.

```
Student john = new Student();
john.initial = 'J';
john.studentId = 1234;
john.enrolmentYear = 2016;
john.courseTaken = new int [2];
john.courseTaken[0] = 2030;
john.courseTaken[1] = 2031;
```

```
public Student (){
    initial = ' ';
    studentId = 0;
    enrolmentYear = 0;
}
```

```
int[] course = {2030, 2031};

Student jade = new Student('J', 5678, course, 2016);
```

V.S.

```
public Student (char init, int sid, int [] cTaken, int eYear) {
    initial = init;
    studentId = sid;
    enrolmentYear = eYear;
    courseTaken = new int [cTaken.length];
    for (int i= 0; i < cTaken.length; i++)
        courseTaken[i] = cTaken[i];
}
```

Which one is preferred and why?

A useful Keyword

- `this`
- `'this'` is a keyword that refers to the instance variables of the current object.

```
public Student (char initial, int studentId, int [] courseTaken, int enrolmentYear) {
   this.initial = initial;
   this.studentId = studentId;
   this.enrolmentYear = enrolmentYear;
   this.courseTaken = new int [courseTaken.length];
   For (int i= 0; i < courseTaken.length; i++)
      this.courseTaken[i] = courseTaken[i];
}
```

How 'this' is replaced.

```
public Student (char initial, int studentId, int [] courseTaken, int enrolmentYear) {
   this.initial = initial;
   this.studentId = studentId;
   this.enrolmentYear = enrolmentYear;
   this.courseTaken = new int [courseTaken.length];
   For (int i= 0; i < courseTaken.length; i++)
      this.courseTaken[i] = courseTaken[i];
}
```

Jade.initial = 'J'
Jade.studentId = 5678

alice.initial = 'a';
alice.studentId = 1256

```
int[] course = {2030, 2031};
Student jade = new Student('J', 5678, course, 2016);
```

```
int [] courseList = {2030, 2031, 2011, 3311};
Student alice = new Student('a', 1256, courseList, 2016);
```

Overloaded Constructors

- To give the user of the class the flexibility on creating an object, multiple version of the constructors can be defined.
- These constructors are different in the number or types of the arguments that they accept.
- They are called `overloaded` constructors.

```
public Student (char initial, int studentId, int enrolmentYear) {
   this.initial = initial;
   this.studentId = studentId;
   this.enrolmentYear = enrolmentYear;
   this.courseTaken = null;
}
```

```
public Student () {
   this.initial = ' ';
   this.studentId = 0;
   this.enrolmentYear = 0;
   this.courseTaken = null;
}
```

```
public Student (char initial, int studentId, int [] courseTaken, int enrolmentYear) {
   this.initial = initial;
   this.studentId = studentId;
   this.enrolmentYear = enrolmentYear;
   this.courseTaken = new int [courseTaken.length];
   for (int i= 0; i < courseTaken.length; i++)
      this.courseTaken[i] = courseTaken[i];
}
```

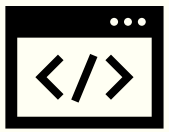
What does null mean?

Constructor Chaining & Overloaded constructors

- To avoid duplicate code, constructors are chained.
- To chain the constructors, `this()` is used.
- `this()` can have zero or more parameters, depending on which constructor is called.

```
public Student (char initial, int studentId, int enrolmentYear) {
        this.initial = initial;
        this.studentId = studentId;
        this.enrolmentYear = enrolmentYear;
}
```

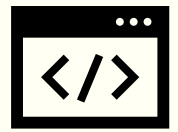
```
public Student (char initial, int studentId, int [] courseTaken, int enrolmentYear) {
        this(initial, studentId, enrolmentYear);
        this.courseTaken = new int [courseTaken.length];
        for (int i= 0; i < courseTaken.length; i++)
                this.courseTaken[i] = courseTaken[i];
}
```



Constructors (3)

- It is possible to clone an object.
- This type of constructors are called `copy constructors`.
- The input parameter is an object of the same type.

```
public Student (Student st) {
        this.initial = st.initial;
        this.studentId = st.studentId;
        this.enrolmentYear = st.enrolmentYear;
}
```



Question

1. Suppose that I have only defined the following constructor. Can I define a new object as `Student alice = new Student(); ?`

```
public Student (char init, int sid, int [] cTaken, int eYear) {
   initial = init;
   studentId = sid;
   enrolmentYear = eYear;
   courseTaken = new int [cTaken.length];
   For (int i= 0; i < cTaken.length; i++)
      courseTaken[i] = cTaken[i];
}
```

2. The input parameters are ugly, can I use more meaningful name such as below?

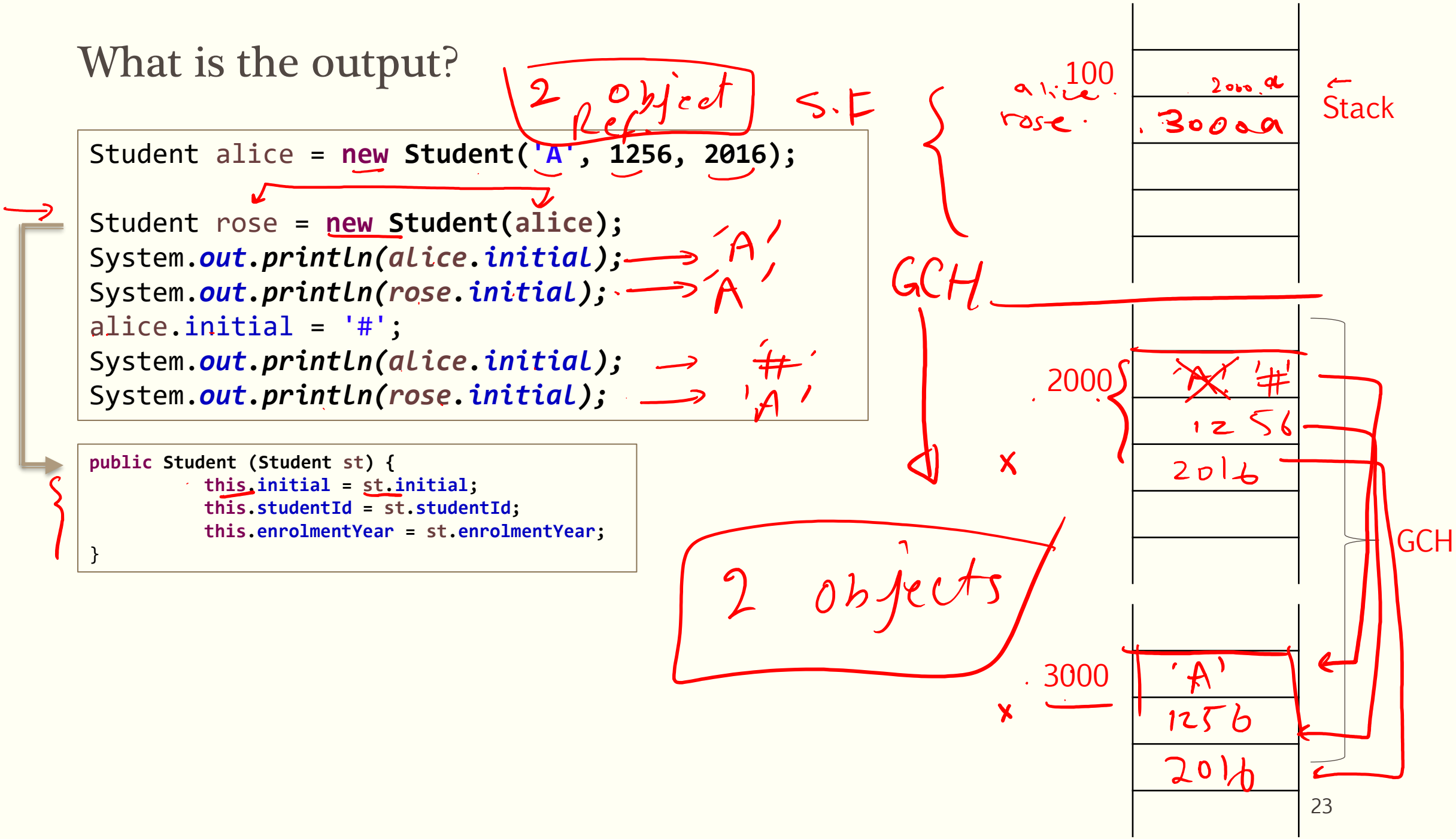
```
public Student (char initial, int studentId, int [] courseTaken, int enrolmentYear) {
   initial = initial;
   studentId = studentId;
   enrolmentYear = enrolmentYear;
   courseTaken = new int [courseTaken.length];
   For (int i= 0; i < courseTaken.length; i++)
      courseTaken[i] = courseTaken[i];
}
```

In-Class Activity 2

What is the output?

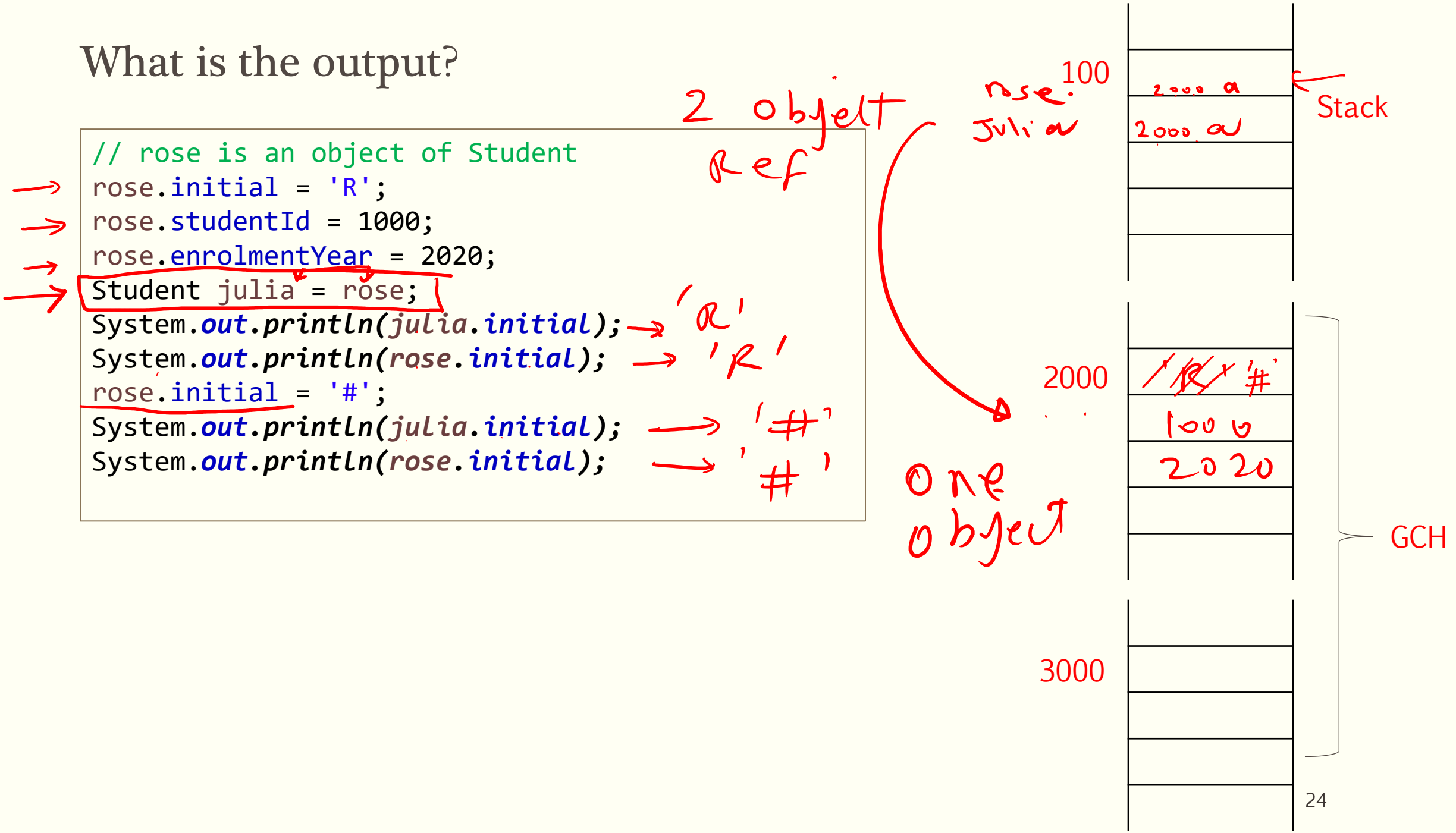
```
Student alice = new Student('A', 1256, 2016);

Student rose = new Student(alice);
System.out.println(alice.initial);
System.out.println(rose.initial);
alice.initial = '#';
System.out.println(alice.initial);
System.out.println(rose.initial);
```

```
public Student (Student st) {
        this.initial = st.initial;
        this.studentId = st.studentId;
        this.enrolmentYear = st.enrolmentYear;
}
```



What is the output?

```
// rose is an object of Student
rose.initial = 'R';
rose.studentId = 1000;
rose.enrolmentYear = 2020;
Student julia = rose;
System.out.println(julia.initial);
System.out.println(rose.initial);
rose.initial = '#';
System.out.println(julia.initial);
System.out.println(rose.initial);
```



Homework

- How do you copy `courseTaken` in the copy constructor so that changing the `courseTaken` for one object does not change its clone's `courseTaken`?

Garbage Collection Mechanism

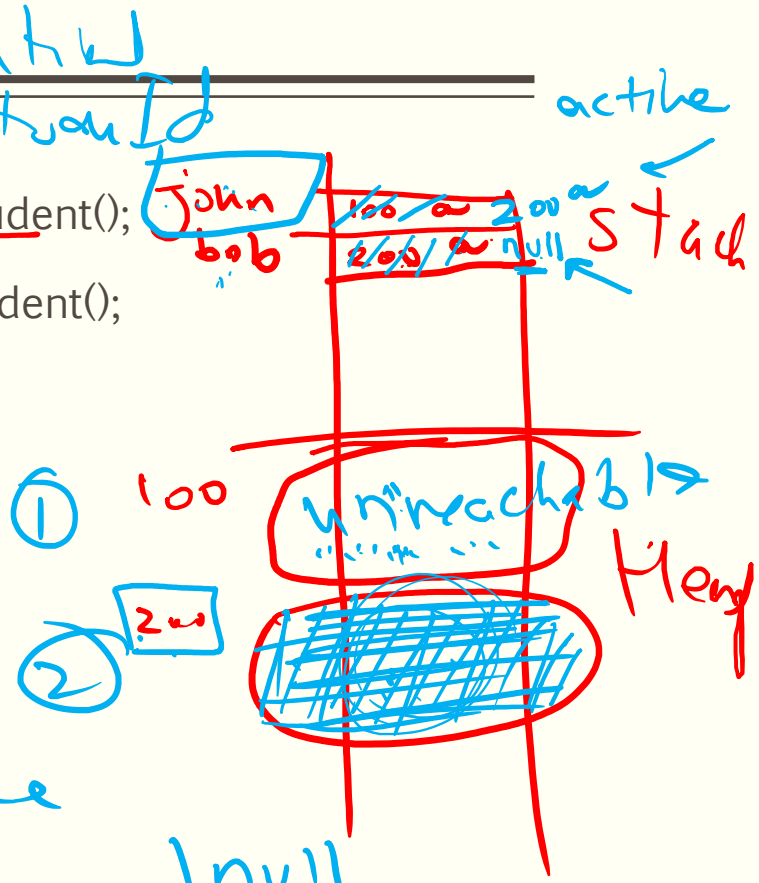
- Objects are stored in the heap.
- If an object is no longer needed, the memory space that is allocated to them should be released.
- This is a process that is performed by Garbage Collection in Java.

object ref. that points to an actual object

- How many active reference, null reference, reachable and unreachable objects do you see in the example?

Obj. Ref. does Not point to any object

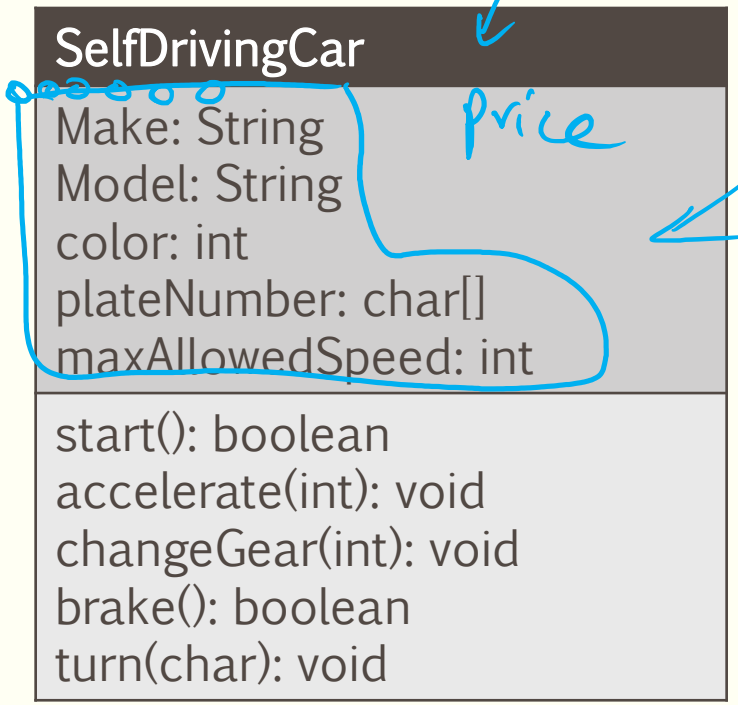
```
Student john = new Student();
Student bob = new Student();

john = bob;

bob = null;
```



In Class Activity 3

Encapsulation

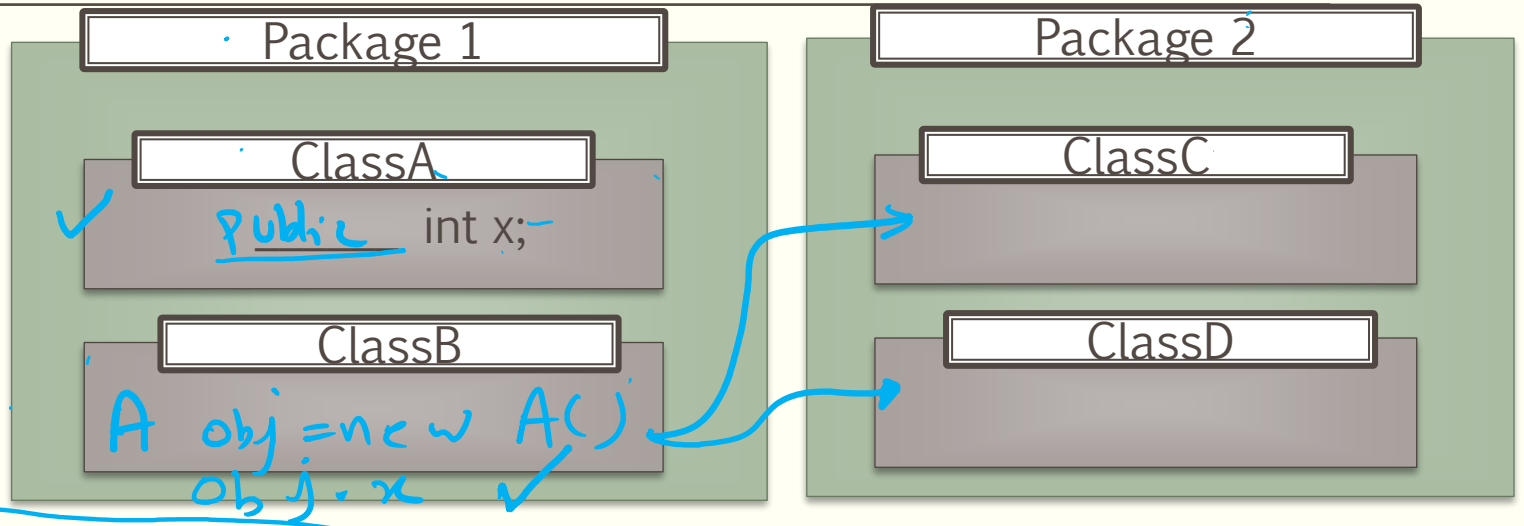
- The first object-oriented principle.
- Data encapsulation refers to `hiding` the internal states of an object
  - an `implementer` controls the `clients`' access to the instance variables.



- The first step is to make the instance variables inaccessible to the clients.
- This is done by choosing right `access modifiers`.

Access Modifiers

- To define an access level for a variable, method or constructors `access modifiers` are used.
- Access modifiers visibility:

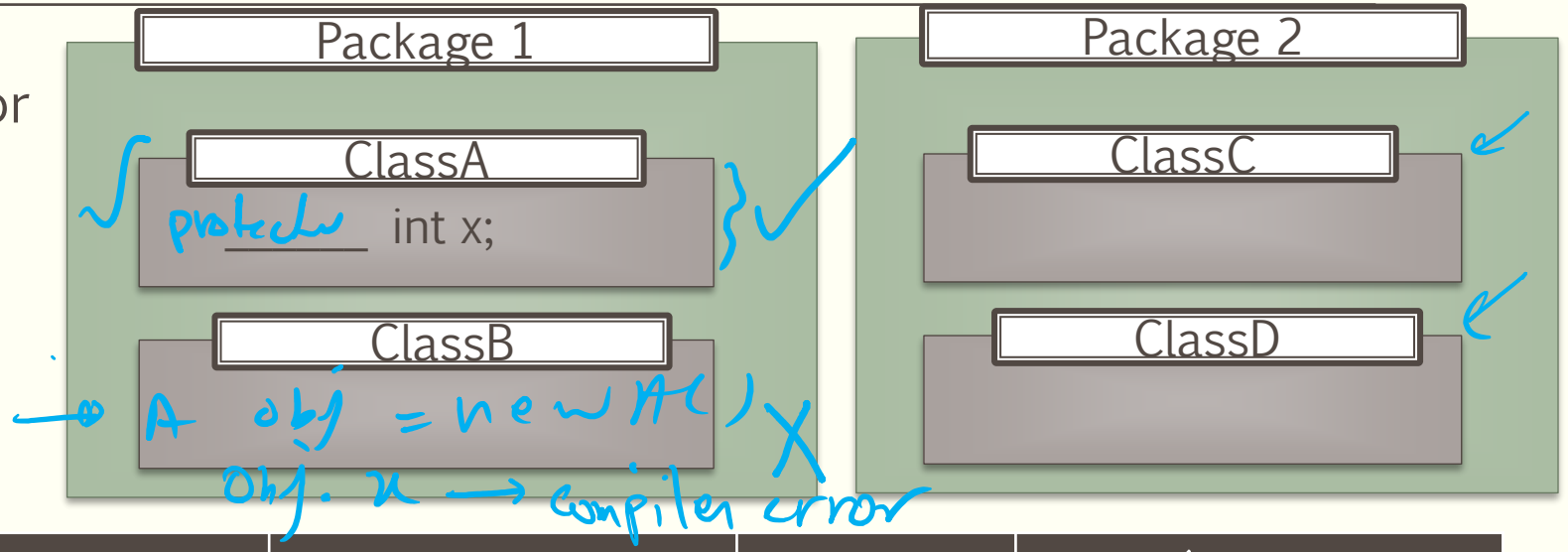


Access Modifier	class	Subclass Same Package	Subclass Outside Package	package	World (outside the package)
public	Y	Y	Y	Y	Y
protected	Y	Y	Y	N	N
No access modifier	Y	Y	N	Y	N
private	Y	N	N	N	N

These two columns will be revisited later

Access Modifiers

- To define an access level for a variable, method or constructors `access modifiers` are used.
- Access modifiers visibility:

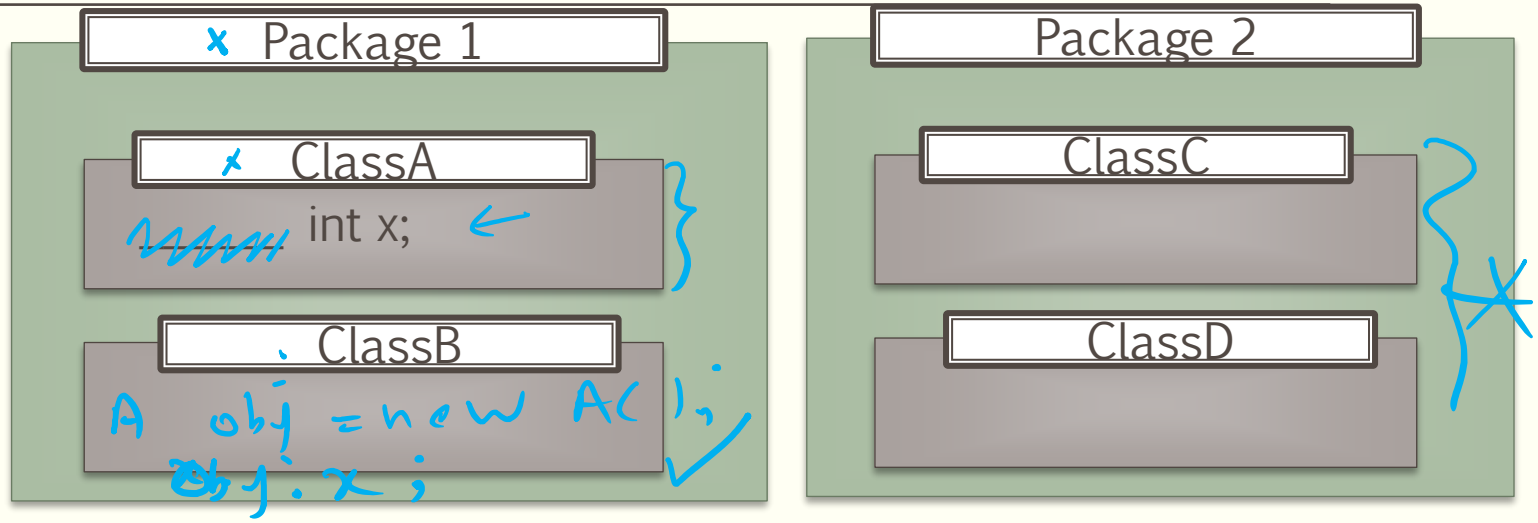


Access Modifier	class	Subclass Same Package	Subclass Outside Package	package	World (outside the package)
public	Y	Y	Y	Y	Y
protected	Y	Y	Y	N	N
No access modifier	Y	Y	N	Y	N
private	Y	N	N	N	N

These two columns will be revisited later

Access Modifiers

- To define an access level for a variable, method or constructors `access modifiers` are used.
- Access modifiers visibility:

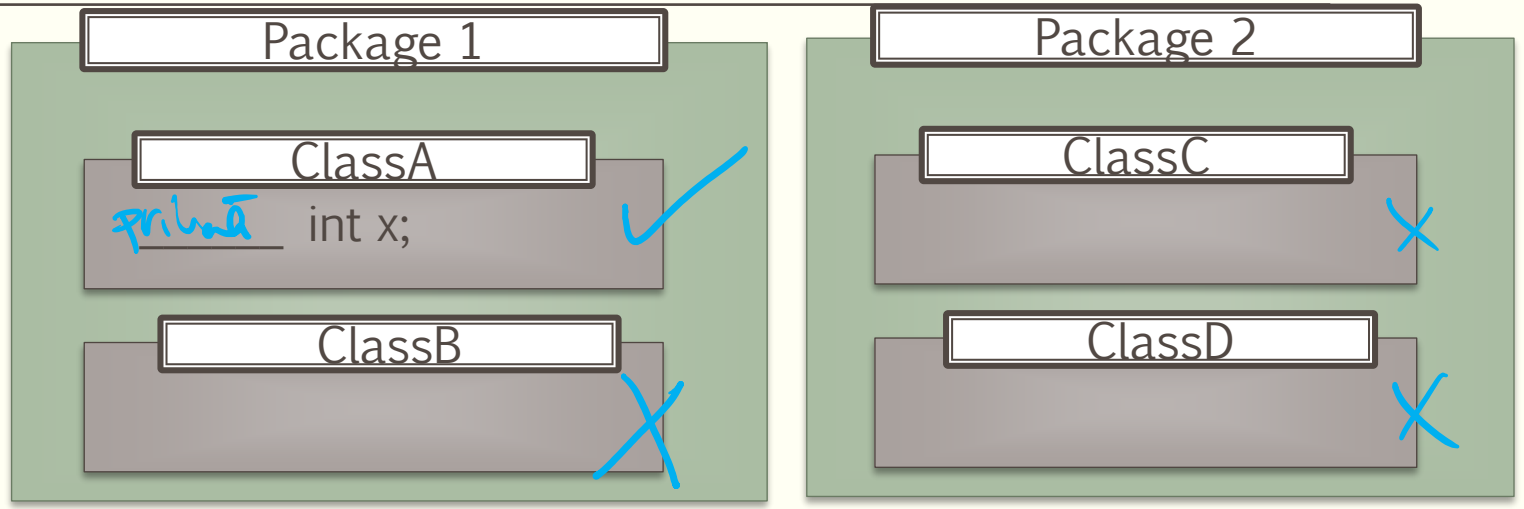


Access Modifier	class	Subclass Same Package	Subclass Outside Package	package	World (outside the package)
public	Y	Y	Y	Y	Y
protected	Y	Y	Y	N	N
No access modifier	Y	Y	N	Y	N
private	Y	N	N	N	N

These two columns will be revisited later

Access Modifiers

- To define an access level for a variable, method or constructors `access modifiers` are used.
- Access modifiers visibility:



Access Modifier	class	Subclass Same Package	Subclass Outside Package	package	World (outside the package)
public	Y	Y	Y	Y	Y
protected	Y	Y	Y	N	N
No access modifier	Y	Y	N	Y	N
private	Y	N	N	N	N

These two columns will be revisited later

Information hiding

```
public class SelfDrivingCar {
    private String make;
    private String model;
    private int color;
    private char[] plateNumber;
    private int maxAllowedSpeed;
}
```

```
public class UseCar {
    public static void main(String [] args) {
        SelfDrivingCar myCar = new SelfDrivingCar();
        myCar.make = "Toyota";
    }

}
```

C.E

NOT ALLOWED

Accessors and Mutators (AKA getters & Setters)

- When data is encapsulated, the only way by which you can get access to the data is via the object methods.
- Accessors (getters): return the value of the instance variable.
  - You can decide in this method, whether or not to provide the access, or how much access to provide.

```
public class SelfDrivingCar {
    private String make;
    private String model;
    private int color;
    private char[] plateNumber;
    private int maxAllowedSpeed;

    public String getMake() {
        return make;
    }
}
```

Naming Convention for getter methods

Accessors and Mutators (AKA getters & Setters)

- When data is encapsulated, the only way by which you can get access to the data is via the object methods.
- Mutators (setters): updates the value of the instance variable.
  - You can check the validity of the value that you assign to the instance variable.

```
public class SelfDrivingCar {
    private String make;
    private String model;
    private int color;
    private char[] plateNumber;
    private int maxAllowedSpeed;

    public String getMake() {
        return make;
    }
    public void setMake(String make) {
        this.make = make;
    }
}
```

Naming Convention for getter methods

Encapsulation at a glance

```java
public class SelfDrivingCar {
    private String make;
    private String model;
    private int color;
    private char[] plateNumber;
    private int maxAllowedSpeed;

    public String getMake() {
        return make;
    }
    public void setMake(String make) {
        this.make = make;
    }
}
```

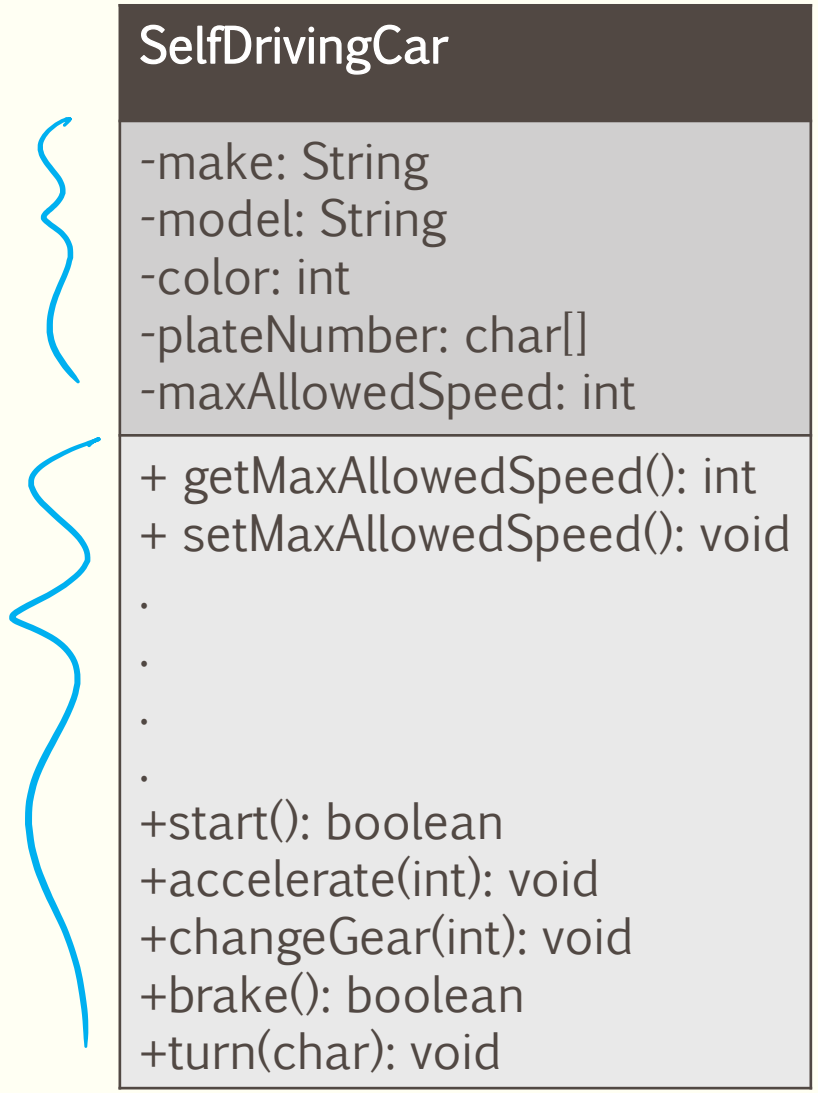
```java
public class UseCar {
    public static void main(String [] args) {
        SelfDrivingCar myCar = new SelfDrivingCar();
        myCar.setMake("Toyota");
        System.out.println(myCar.getMake());
    }

}
```

How to recognize which instance variable should be private?

- There is no correct answer as to what instance variables should be public. It is application specific.
- Rule of thumb: if you are not sure how sensitive a data is, make the variable invisible and provide the access by mutator and accessor methods.

UML Presentation

- Public members are shown by +.
- Private members are shown by -.



Homework

- Use memory model to identify what will be printed.

```
public class SelfDrivingCar {
    private String make;
    private String model;
    private int color;
    private char[] plateNumber;
    private int maxAllowedSpeed;

    public String getMake() {
        return make;
    }
    public void setMake(String make) {
        this.make = make;
    }

    public char [] getPlateNumber() {
        return plateNumber;
    }
    public void setPlateNumber(char[] plateNumber) {
        this.plateNumber = plateNumber;
    }
}
```

```
public class UseCar {
    public static void main(String [] args) {
        SelfDrivingCar myCar = new SelfDrivingCar();
        myCar.setMake("Toyota");
        char [] pno = {'x','x','x','x','1','1','1'};
        myCar.setPlateNumber(pno);
        pno[0]='s';
        System.out.println(myCar.getPlateNumber());
    }
}
```

- This is a security whole. How do you repair it?

Why OOP?

- Bundling code into individual software objects provides a number of benefits, including:
  - Modularity
  - Information-hiding
  - Code re-use
  - Pluggability and debugging ease

Expectations & Reading

- Expectations:
  - You should be able to demonstrate the status of the memory for primitive and non-primitive variables.
  - You should understand what the constructor is and how it is implemented.
  - You should be able to implement chained constructors.
  - You should be able to explain what is the garbage collector in Java
  - You should be able to implement encapsulation.

- Reading: while not necessary, I encourage you to read
  - https://www.w3schools.com/java/java_oop.asp
  - https://www.w3schools.com/java/java_classes.asp
  - https://www.w3schools.com/java/java_class_attributes.asp
  - https://www.w3schools.com/java/java_class_methods.asp
  - https://www.w3schools.com/java/java_constructors.asp
  - https://www.tutorialspoint.com/java/java_access_modifiers.htm
  - https://www.tutorialspoint.com/java/java_encapsulation.htm
- One Minute Paper