22nd Feb 2024

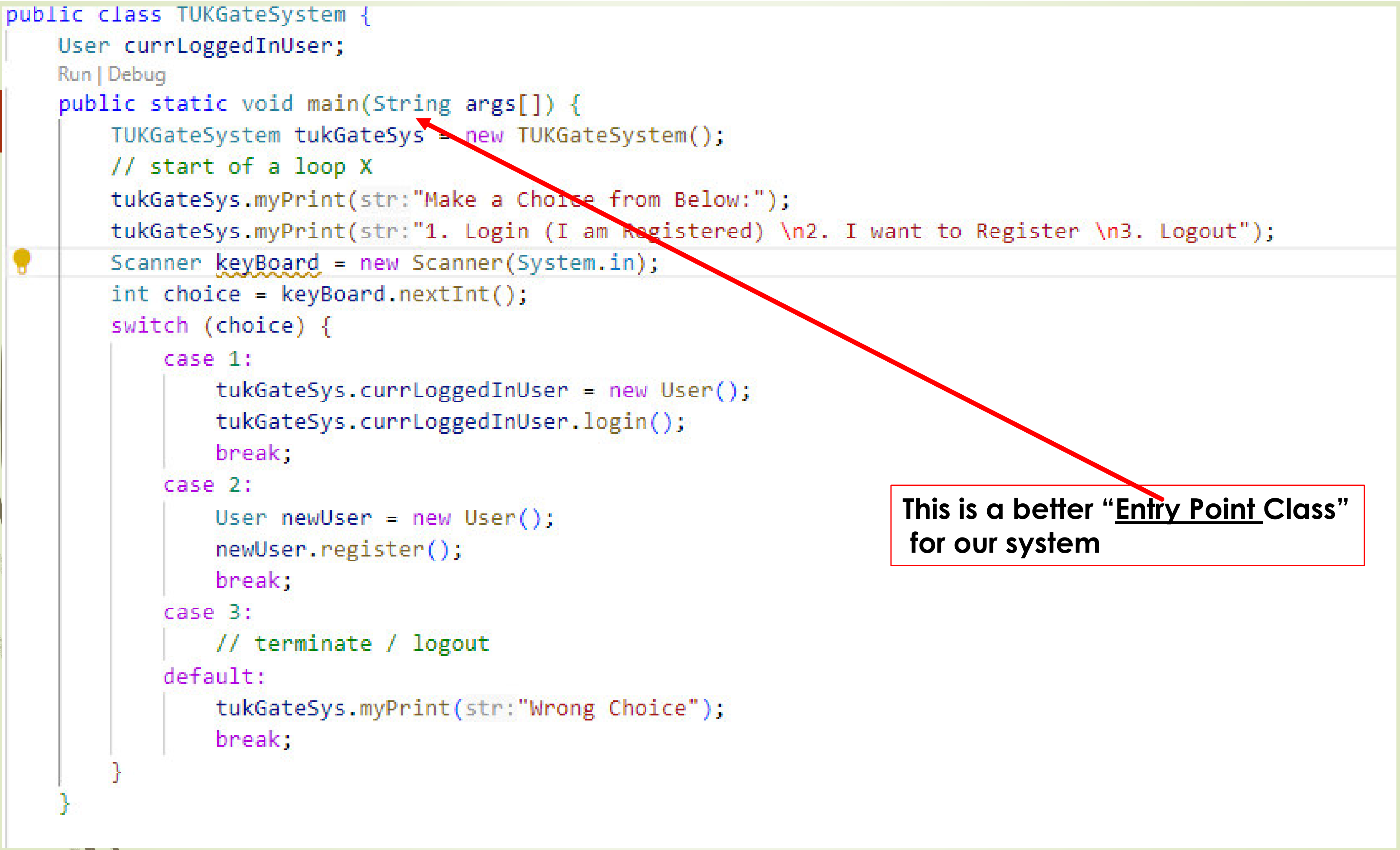
constructors, inheritance, methods, Overloading, overriding, Polymorphism, etc.

```
public class User{

    String name;
    String userName, password;
    int identification;

    public static void main (String args [])
    {
     System.out.println ("This is a User Class");

    }
}
```

Remember:
This is a sample
User class

Makes this class an Application, meaning it can "execute/run"

Not Necessary!



```
public class TUKGateSystem {
    User currLoggedInUser;
    Run | Debug
    public static void main(String args[]) {
        TUKGateSystem tukGateSys = new TUKGateSystem();
        // start of a loop X
        tukGateSys.myPrint(str:"Make a Choice from Below:");
        tukGateSys.myPrint(str:"1. Login (I am Registered) \n2. I want to Register \n3. Logout");
        Scanner keyBoard = new Scanner(System.in);
        int choice = keyBoard.nextInt();
        switch (choice) {
            case 1:
                tukGateSys.currLoggedInUser = new User();
                tukGateSys.currLoggedInUser.login();
                break;
            case 2:
                User newUser = new User();
                newUser.register();
                break;
            case 3:
                // terminate / logout
            default:
                tukGateSys.myPrint(str:"Wrong Choice");
                break;
        }
    }
```

This is a better "**Entry Point** Class" for our system

Practical Work!!

Plenary and individual work

- Create a HIS system (folder)
  - Ref the assignment (Earlier shared case scenarios in assignment 1)
  -
- *twende kazi !*

Inheritance demo

- Create child class using **extends** keyword
  - Admin user
  - Normal User
    - BOTH can, without any code, print the login message, print their details

Add functionalities to the classes

User Class

- name
- userName
- password
- Identification
- type
- login()

SystemAdmin Class

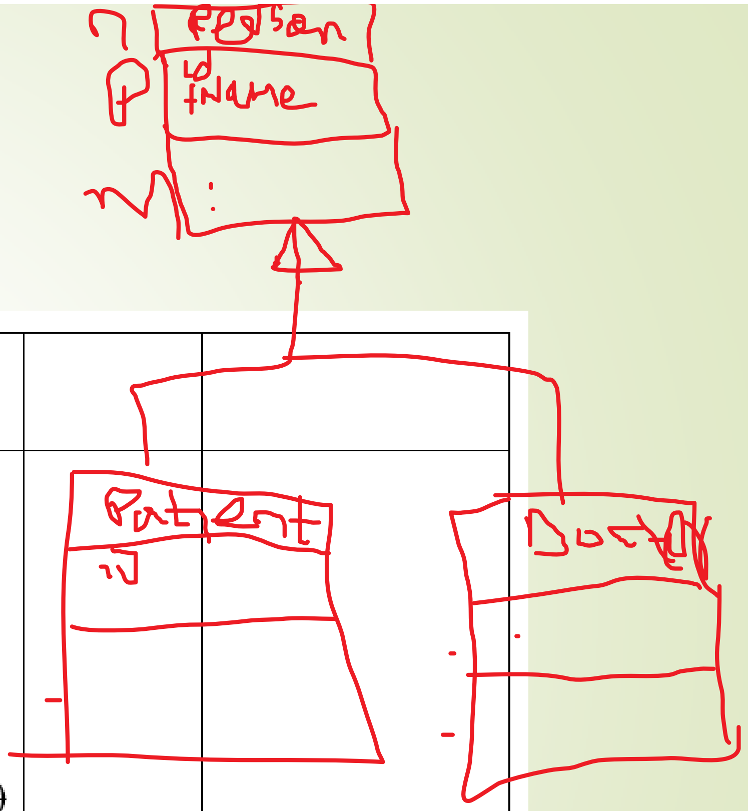
- **name**
- ~~**userName**~~
- ~~**password**~~
- ~~**login()**~~
- ~~**Type::**~~ superUser
- verify()
- listVerifiedUsers()

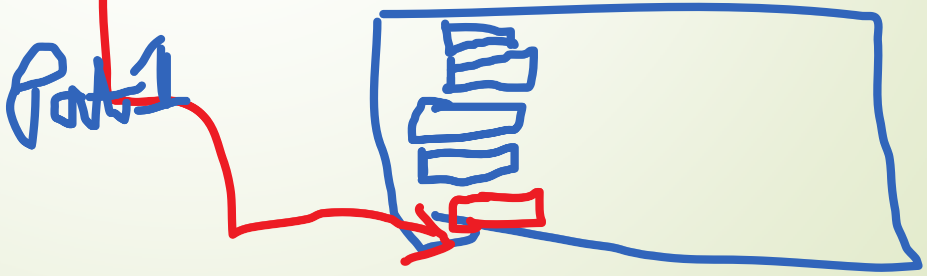
NormalUser Class

- ~~**name**~~
- ~~**userName**~~
- ~~**password**~~
- ~~**login()**~~
- ~~**Type::**~~ normal
- attendClass()
- specifyReason()

- We revisit the **flow chart** /Algorithm of the APP
  - Get user input :: select what a user wants to do
  - Register:
    - Prompt for details
    - …
  - Login
    - Prompt for a username and password

Person	Patient	Doctor	Nurse		
- nationalId - fName - sName - address - gender printDetails()	~~- nationalId~~ - patientId ~~- fName~~ ~~- sName~~ ~~- address~~ ~~- gender~~ ~~printDetails()~~	~~- nationalId~~ - doctorId ~~- fName~~ ~~- sName~~ ~~- address~~ ~~- gender~~ ~~printDetails()~~	~~- nationalId~~ - nurseId ~~- fName~~ ~~- sName~~ ~~- address~~ ~~- gender~~ ~~printDetails()~~		

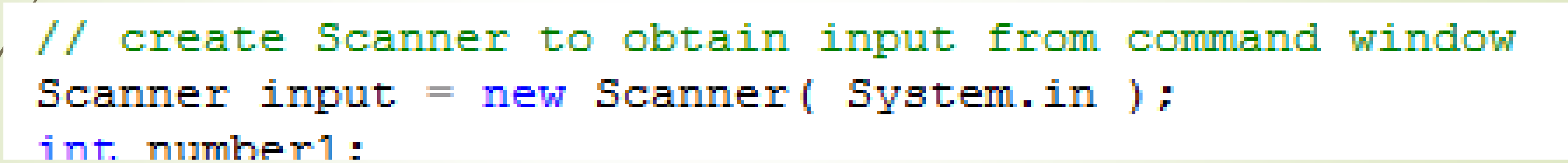




- What are objects?
  - an object represents an individual, identifiable item, unit, or entity, either real or abstract, with a well-defined role in the problem domain.
  - An "object" is anything to which a concept applies

- We have identified and implemented classes that represent Objects in the scenario/group project assignment earlier

- How are objects created?
  - A class ***constructor*** is used to create instances of a class by using the ***new*** keyword
  - Recall



```
// create Scanner to obtain input from command window
Scanner input = new Scanner( System.in );
int number1;
```

- Class constructor
  - **Constructor name is class name**. A constructor **must** have the *same name as the class its in*
  - **Default constructor**. If you don't define a constructor for a class, a *default parameterless constructor* is automatically created by the compiler. The default constructor calls the default parent constructor (super() from the Object class) and initializes all instance variables to default value (zero for numeric types, null for object references, and false for booleans).

- **Default constructor is created only if there are no constructors**. If you define *any* constructor for your class, no default constructor is automatically created.
- **Differences between methods and constructors**. There is *no return type* given in a constructor signature (header). The value is this object itself so there is no need to indicate a return value.
  - There is *no return statement* in the body of the constructor.
  - The *first line* of a constructor must either be a call on another constructor in the same class (using **this**), or a call on the superclass constructor (using super). If the first line is neither of these, the compiler automatically inserts a call to the parameterless super class constructor.

Example – user class Constructor

- User class constructor
  - Register() method in user class :: creates an instance
- Admin User Constructor
  - Updating a field that only Admin has
- Normal User constructor

How are you?

```
package tuk;
/**
 * @author Salesio
 */
public class TUKGateSystem{
    /**
     This class is the Application entry
       point*/
    public static void main(String[] args) {
        // TODO code application logic here
        System.out.println("I am the main method");
        Person person1 = new Person(2,"Salesio");
        System.out.println(person1.password);
    }
}
```

```
package tuk;
// @author Salesio
public class Person {
    int id;
    String name;
    String password;
    public Person(int anId, String aName){
        id=anId;
        name = aName;
      }
}
```

What is the Output?

What is the Output, now?

```
package tuk;
/**
 * @author Salesio
 */
public class TUKGateSystem{
   /**
    This class is the Application entry
      point*/
   public static void main(String[] args) {
      // TODO code application logic here
      System.out.println("I am the main method");
      Person person1 = new Person(2,"Salesio");
      System.out.println(person1.password);
   }
}
```

```
package tuk;
// @author Salesio
public class Person {
   int id;
   String name;
   String password;
   public Person(int anId, String aName){
      id=anId;
      name = aName;
password = Security().createPassword(id, name);
    } }
```

```
package tuk;
/** @author Salesio */
class Security {
    String createPassword(int id,
String name) {
        String pass =
id+name.substring(1);
        return pass;
    }
}
```

- Why would having two or more classes above make sense?
  - Easy maintenance!
  - E.g. assume that the rules of generating usernames change. We only have to change and compile the class that generates Ids/Usernames

Overriding

- ? Is over?

Overloading

- ?

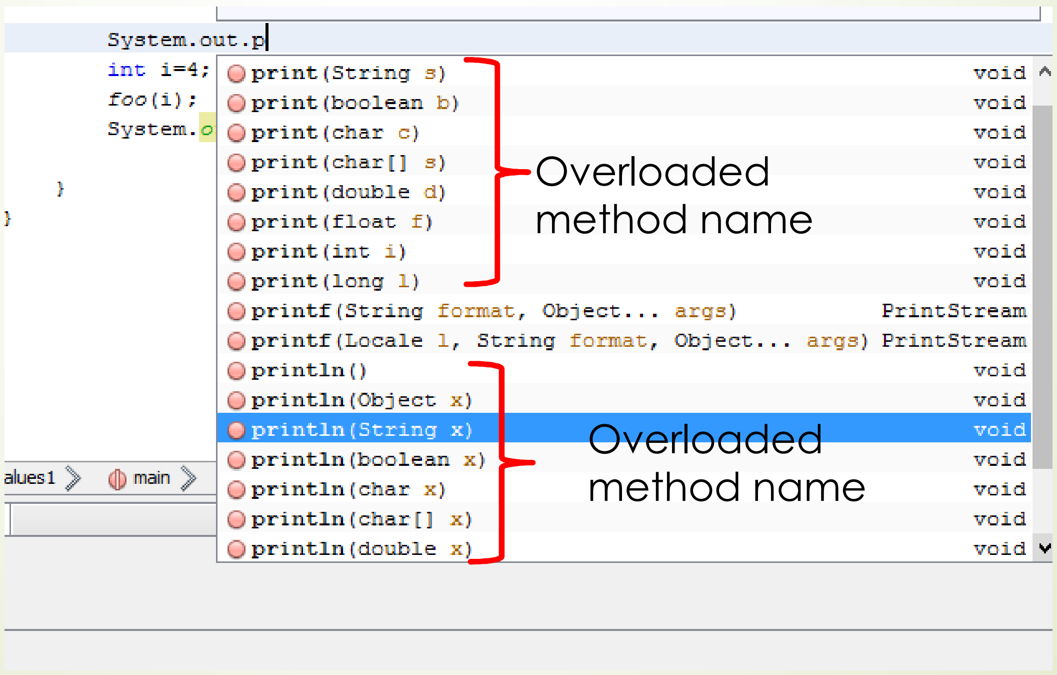
Overloading Methods

- Two methods have the same name but different parameter lists within one class. The Java compiler determines which method is used based on the method signature.

```
  public static double max(double num1, double num2) {
           if (num1 > num2)
           return num1;
           else
           return num2;
     }
```

```
public static int max(int num1, int num2) {
        int result;
        if (num1 > num2)
            result = num1;
        else
            result = num2;
        return result;
    }

public static double max(double num1, double num2) {
        if (num1 > num2)
                return num1;
            else
                return num2;
        }
```

- If you call max with int parameters, the max method that expects int parameters will be invoked; if you call max with double parameters, the max method that expects double parameters will be invoked.



```
System.out.p
int i=4;
foo(i);
System.o
    }
}
```

```
print(String s)                                     void
print(boolean b)                                    void
print(char c)                                       void
print(char[] s)                                     void
print(double d)                                     void
print(float f)                                      void
print(int i)                                        void
print(long l)                                       void
printf(String format, Object... args)        PrintStream
printf(Locale l, String format, Object... args) PrintStream
println()                                           void
println(Object x)                                   void
println(String x)                                   void
println(boolean x)                                  void
println(char x)                                     void
println(char[] x)                                   void
println(double x)                                   void
```

Overriding methods

- Definition:
  - In any object-oriented programming language, Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes.
  - When a method in a subclass has the same ***name***, same ***parameters*** or **signature**, and same **return type**(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.
- Uses
  - to provide the specific implementation of a method which is already provided by its superclass
  - To implement runtime polymorphism - If an object of a parent class is used to invoke the method, then the version in the parent class will be executed, but if an object of the subclass is used to invoke the method, then the version in the child class will be executed
- REF:
  - https://pragjyotishcollege.ac.in/wp-content/uploads/2020/04/Method-Overriding-in-Java.pdf

- Implementation examples
  - Ideas
    -

- Thanks

Polymorphism

Polymorphism

- "poly" means many and "morphs" means forms. So polymorphism means many forms.
- Polymorphism is not a programming concept but it is one of the principal concepts of OOPs.
- Polymorphism is common in OOP languages but is implemented differently

Dynamic or runtime polymorphism

- Is achieved via method overriding (in inheritance)
- Reference of *parentclass* (e.g. Person) can hold object of the *parentclass* (i.e Person) or an object of any of its subclass (e.g. Staff)
- E.g. the following statements are valid:
  - Person obj = new Staff(); // parent class reference can be assigned to child object
- Since in method overriding both the classes(parent class and child class) have same method, compiler doesn't figure out which method to call at compile-time. In this case JVM(java virtual machine) decides which method to call at runtime that's why it is known as runtime or dynamic polymorphism.

Example

```
public class Person {
   int id;
   String name;
   public Person(int anId, String aName){
      id=anId;
      name = aName;
     }
  public void printDetails(){
     System.out.println("ID: "+id);
     System.out.println("Name: "+name);
    }
}
```

```
public class Staff extends Person{
   String staffId;
   Staff(String aStaffId, String staffName,
               int nationalId)
    {
      super(nationalId, staffName);
      this.staffId = aStaffId;
    }
   public void printDetails(){
      super.printDetails();
      System.out.println("StaffPayroll: “
          +staffId);
      }
}
```

```
public class HIS {
public static void main(String[] args) {
Person pers1 = new Person(12345,"Haile");
      Person pers2 = new Staff("FA2125", “Jonathan", 123234);
      pers2.printDetails();
      System.out.println("--------------------");
      pers1.printDetails();
   }
}
```

Static or Compile time Polymorphism

- This is achieved via method overloading
- A class has more than one methods with the same methods ‘***sharing***’ the same name but different number of arguments or different types of arguments or both
- When the method is called on an object, it is the parameters used in the call that determines what method code is executed

```
class X
{
   void methodA(int num) {
       System.out.println ("methodA:" + num);
   }
   void methodA(int num1, int num2){
       System.out.println ("methodA:" + num1 + ","
+ num2);
   }
   double methodA(double num) {
       System.out.println("methodA:" + num);
       return num;
   }
}
```

```
class Y{
   public static void main (String args []){
       X Obj = new X();
       double result;
       Obj.methodA(20);
       Obj.methodA(20, 30);
       result = Obj.methodA(5.5);
       System.out.println("Answer is:" + result);
   }
}
```

Output?:

```
class X
{
  void methodA(int num) {
     System.out.println ("methodA:" + num);
  }
  void methodA(int num1, int num2){
     System.out.println ("methodA:" + num1 + ","
+ num2);
  }
  double methodA(double num) {
     System.out.println("methodA:" + num);
     return num;
  }
}
```

```
class Y{
  public static void main (String args []){
     X Obj = new X();
     double result;
     Obj.methodA(20);
     Obj.methodA(20, 30);
     result = Obj.methodA(5.5);
     System.out.println("Answer is:" + result);
  }
}
```

Output (answer):

```
methodA:
methodA:
methodA:
Answer is:
```

```
class X
{
  void methodA(int num) {
     System.out.println ("methodA:" + num);
  }
  void methodA(int num1, int num2){
     System.out.println ("methodA:" + num1 + ","
+ num2);
  }
  double methodA(double num) {
     System.out.println("methodA:" + num);
     return num;
  }
}
```

```
class Y{
  public static void main (String args []){
     X Obj = new X();
     double result;
     Obj.methodA(20);
     Obj.methodA(20, 30);
     result = Obj.methodA(5.5);
     System.out.println("Answer is:" + result);
  }
}
```

Output (answer):

```
methodA:20
methodA:20,30
methodA:5.5
Answer is:5.5
```

Do we have Polymorphism here? Explain.

Next

- Objects and Classes (personal work implementation
- Methods, Java parameter passing
- mini project (initial Assessment)

- Thank you