Tab 1

Assignment 1

1. **“Have you ever felt significantly impeded by bad code? If you are a programmer of any experience, then you’ve felt this impediment many times. Indeed, we have a name for it. We call it *wading*.” (pg. 3, chp. 1)**
    - This quote resonated with me because I have suffered through this thing so many times, and I had no idea what to call it. Back in my freshman year, I STRUGGLED with figuring out certain bits of code whenever the program didn't work. It would take me hours just to figure out how the program was breaking, and most of the time, it was because of something small, like a function not being ended correctly, or a line of code not going to a specific class. Whenever I find out about this, it just leaves me there like: “god, why was that the reason you were being a pain?”

2. **//Checking_Account//**

```
import java.util.Scanner;

public class CheckingAccount implements HasMenu{

   double balance;


   public String menu(){
      Scanner input = new Scanner(System.in);
      System.out.println("0) Exit");
      System.out.println("1) Check balance");
      System.out.println("2) Make a deposit");
      System.out.println("3) Make a withdrawal");

      System.out.println();
      System.out.println("Please enter 0-3: ");

      String result = input.nextLine();
      return result;

}

  public void start(){
    boolean keepGoing = true;
                while (keepGoing){
                        String result = menu();
                        if (result.equals("0")){
                                keepGoing = false;
                        } else if (result.equals("1")){
                                getBalanceString();
                        } else if (result.equals("2")){
                        makeDeposit();
                } else if (result.equals("3")){
          makeWithdrawal();
    } else {
                        System.out.println("Please enter 0-2");
                        }
                }
}

public double getBalance(){
   return this.balance;
}

public String getBalanceString(){
   String result = String.format("$%f", balance);
   return result;
}

public void setBalance(double balance){
   this.balance = balance;
}

public void checkBalance(){
   System.out.println("Current balance: " + this.getBalanceString());
}
```

```
    public void makeDeposit(){
        Scanner input = new Scanner(System.in);

        System.out.println("Making a deposit...");
        System.out.println("How much do you want to deposit?: ");
        Double depositResult = input.nextDouble();
        this.balance += depositResult;
    }

    public void makeWithdrawal(){
        Scanner input = new Scanner(System.in);

        System.out.println("Making a withdrawal...");
        System.out.println("How much do you want to withdrawal?: ");
        Double withdrawalResult = input.nextDouble();
    }

}
```

3. Classes and objects should have noun or noun phrase name like “Customer,” “WikiPage,” “Account,” and “AddressParser.” Avoid words like “Manager,” “Processor,” “Data,” “Info” in the name of a class. A class name should be a verb.

Tab 2

Qaden Childs - Assignment 2

1. https://github.com/Dung30ns-Dr0g0ns546/LeapYearCalculator
2. //Java Guesser

```
import java.util.Random;
import java.util.Scanner;


public class Guesser {
   Scanner input = new Scanner(System.in);
   Random random = new Random();
   int randomNumber = random.nextInt(100) + 1;
   int attempt = 0;

   public static void main(String[] args) {
      new Guesser();
   }


   public Guesser(){
      boolean keepGoing = true;
      while(keepGoing){
         String result = menu();
         if (result.equals("0")){
            keepGoing = false;
         } else if (result.equals("1")){
            humanGuesser();
         } else if (result.equals("2")){
            computerGuesser();
         } else {
            System.out.println("Please enter 0-2");
         }

      };
}

public String menu(){
   Scanner input = new Scanner(System.in);

   System.out.println("0). Exit");
   System.out.println("1). Human Guesser");
   System.out.println("2). Computer Guesser");

   System.out.println();
   System.out.println("Please enter 0-2: ");

   String result = input.nextLine();
   return result;
}

public void humanGuesser(){
   Scanner input = new Scanner(System.in);
   int attempts = 0;
   System.out.println("I am thinking of a number between 1 and 100. Try your best to guess it.");

   boolean keepGoing = true;
   while(keepGoing){

      if(attempts <= 7){

         System.out.printf("%d). Please guess the number: ",attempts);
         int userResult = input.nextInt();
         attempts++;

         if (userResult > randomNumber){
            System.out.println("Too high, try again.");
         } else if (userResult < randomNumber){
            System.out.println("Too low, try again.");
         } else if (userResult == randomNumber){
            System.out.println("Congrats! You guessed the number!");
```

```
            keepGoing = false;
        } else {
            System.out.println("You've ran out of tries. Better luck next time...");
            keepGoing = false;
        }
    }else{
        System.out.println("You lose!");
        keepGoing = false;
    }
  }

}

public void computerGuesser(){
   System.out.println("Think of a number between 1 and 100. I'm going to try and guess it.");
   boolean keepGoing = true;
   int low = 0;
   int high = 100;
   int guess = (low + high) / 2;


   while(keepGoing){
       attempt++;

       System.out.printf("%d). My guess is %d was higher(h), lower(l), or correct(c)?: \n",attempt,guess);
       String response = input.nextLine();


       if (response.equals("c")){
          System.out.println("Yay! I guessed it in " + attempt + " attempts.");
            keepGoing = false;
          } else if (response.equals("h")) {
            low = guess + 1;
          } else if (response.equals("l")) {
            high = guess - 1;
          } else {
            System.out.println("Please respond with h(higher), l(lower), c(correct).");
          }

          guess = (low + high) / 2;

          if (low > high) {
            System.out.println("Hmm, it seems there was a misunderstanding. Let's try again.");
            keepGoing = false;
          }
      }
   }
}
```

3. **Important Principle: A function in a class should be small, do one thing, and should use descriptive names**
4. **I made this code in September 25 of last year**

   (https://docs.google.com/document/d/1cf0a3RvETB2U5AnXMHjQmgXjK1bMmUwChpr8D2bSsWM/edit?tab=t.wklxqjkdzv03#heading=h.2rhfiwp52fc6)

Tab 3

Qaden Childs - Assignment 3

1. https://github.com/Dung30ns-Dr0g0ns546/SillyMethods
2. Whenever I write a comment, it should explain why I am making this piece of code, not what the code is. Additionally, being consistent with my formatting will help make my code readable and understandable, thereby reducing the cognitive load required to read and maintain it.

Tab 4

Qaden Childs - Assignment 4

1. In the video, the topic of inheritance, encapsulation, and polymorphism is stated at 10:16
2. public class Bank implements HasMenu {

```
	Admin admin = new Admin();

	private ArrayList<Customer> customers = new ArrayList<>();


	public static void main(String[] args){
		new Bank();
	}

	public Bank(){
		//this.loadSampleCustomers();
		//this.saveCustomers();
		this.loadCustomers();
		this.start();
		this.saveCustomers();

	}

   public void saveCustomers(Customer customer){
		customers.add(customer);
System.out.println("Customer saved: " + customer.getUserName());
   }

   public void loadCustomers(){
		customers.add(new Customer("Alice", "1234"));
		customers.add(new Customer("Bob", "5678"));
		customers.add(new Customer("Cindy", "2468"));

   }

   public void fullCustomerReport(){
		if (customers.isEmpty()) {
   System.out.println("No customers found.");
} else {
   for (Customer customer : customers) {
      System.out.println(customer);
   }
}
   }

   public void addUser(){
		if (customers != null) {
			customers.add(customers);
			System.out.println("User " + user.getUserName() + " added successfully.");
		} else {
			System.out.println("Invalid user data. Cannot add to bank.");
		}
}

      public void applyInterest(){
			//For loop that goes through every customer and applys interest to it's savings account
			double interest = balance * interestRate;
   balance += interest;
      }

      public String menu(){
			Scanner input = new Scanner(System.in);

			System.out.println("0) Exit System");
   System.out.println("1) Login as Admin");
   System.out.println("2) Login as Customer");

   System.out.println();
   System.out.println("Please enter 0-2: ");

   String result = input.nextLine();
```

```
return result;
    }

    public void loginAsCustomer(){
            Scanner input = new Scanner(System.in);

            System.out.print("User Name: ");
            String userNameIn = input.nextLine();
            System.out.print("PIN: ");
            String pinIn = input.nextLine();

            Customer currentCustomer = null;
            for (Customer customer: customers){
                    if (customer.login(userNameIn, pinIn)){
                            currentCustomer = customer;
                    }
            }

            if (currentCustomer == null){
                    System.out.println();
            }
    }

public void start(){
        boolean keepGoing = true;
        while (keepGoing){
                String result = menu();
                if (result.equals("0")){
                        keepGoing = false;
                } else if (result.equals("1")){
                        if (admin = true);
                        startAdmin();
                } else if (result.equals("2")){
                loginAsCustomer();
        } else {
                System.out.println("Please enter 0-2");
                }
        }
}

public void startAdmin(){
        boolean keepGoing = true;
        while (keepGoing){
                String response = admin.menu();
                if (response.equals("0")){
                        keepGoing = false;
                } else if(response.equals("1")){
                        System.out.println("Full Customer Report");
                        this.fullCustomerReport();
                } else if (response.equals("2")){
                        System.out.println("Add user");
                        this.addUser();
                } else if (response.equals("3")){
                        System.out.println("Apply Interest");
                        this.applyInterest();
                } else {
                        System.out.println("Please enter 0-3");
                }
        }
    }

}

class Customer extends User{

        CheckingAccount checking = new CheckingAccount();
        SavingsAccount savings = new SavingsAccount();

        public Customer(){
                this.userName = "John Doe";
                this.pin = "1234";
        }
```

```
public Customer(String usernName, String PIN){
	this.userName = userName;
	this.pin = pin;
}

public void start(){
	boolean keepGoing = true;
	while (keepGoing){
		String result = menu();
		if (result.equals("0")){
			keepGoing = false;
		} else if (result.equals("1")){
			checking = new CheckingAccount();
		} else if (result.equals("2")){
			savings = new SavingsAccount();
		} else if (result.equals("3")){
			changePin();
		} else {
			System.out.println("Please enter 0-3");
		}
	}
   }

   public String menu(){
	Scanner input = new Scanner(System.in);

	System.out.println("0) Exit");
System.out.println("1) Manage Checking Account");
System.out.println("2) Manage Saving Account");
	System.out.println("3) Change PIN");

System.out.println();
System.out.println("Please enter 0-3: ");

String result = input.nextLine();
return result;
   }

   public void changePin(){
	Scanner input = new Scanner(System.in);

	System.out.println("Enter New PIN: ");

	String newPin = input.nextLine();

	if (newPin.equals(pin)){
		System.out.println("Incorrect. You have entered your current PIN. Try again.");
	} else {
			System.out.println("New PIN has been set.");
		}
	  }

	  public String getReport(){
	return String.format("UserName: %s\nChecking: %s\nSaving: %s",userName, checking, savings);
	  }
}

public class Admin extends User implements HasMenu , Serializable{

    public String menu(){
			Scanner input = new Scanner(System.in);

	System.out.println("0) Exit This Menus");
	System.out.println("1) Full Customer Report");
	System.out.println("2) Add User");
	System.out.println("3) Apply Interest To Savings Accounts");

	System.out.println();
	System.out.println("Please enter 0-3: ");
```

```
        String result = input.nextLine();
        return result;
    }

    public void start(){

    }




    public String getReport(){
        return String.format("UserName: %s\nPIN: %s\n",userName,pin);
    }
}

class SavingsAccount extends CheckingAccount{

    private double interestRate;

            public String menu(){
        Scanner input = new Scanner(System.in);
        System.out.println("0) Exit");
        System.out.println("1) Check balance");
        System.out.println("2) Make a deposit");
        System.out.println("3) Make a withdrawal");

        System.out.println();
        System.out.println("Please enter 0-3: ");

        String result = input.nextLine();
        return result;

}

  public void start(){

}

        public static void main(String[] args){
                new SavingsAccount();
        }

        public void calcInterest(){
                double interest = this.balance * this.interestRate;
    this.balance += interest;
     System.out.println("Interest calculated and added. New balance: " + this.balance);
        }
            public void setInterestRate(double interestRate){
        this.interestRate = interestRate;
            }

            public double getInterestRate(){
        return this.interestRate;
            }
}

public class CheckingAccount implements HasMenu{

    double balance;


    public String menu(){
        Scanner input = new Scanner(System.in);

        System.out.println("0) Exit");
        System.out.println("1) Check balance");
        System.out.println("2) Make a deposit");
        System.out.println("3) Make a withdrawal");
```

```
    System.out.println();
    System.out.println("Please enter 0-3: ");

    String result = input.nextLine();
    return result;

}

  public void start(){
    boolean keepGoing = true;
                while (keepGoing){
                        String result = menu();
                        if (result.equals("0")){
                                keepGoing = false;
                        } else if (result.equals("1")){
                                getBalanceString();
                        } else if (result.equals("2")){
                        makeDeposit();
                } else if (result.equals("3")){
       makeWithdrawal();
    } else {
                        System.out.println("Please enter 0-2");
                        }
                }
}

public double getBalance(){
    return this.balance;
}

public String getBalanceString(){
    String result = String.format("$%f", balance);
    return result;
}

public void setBalance(double balance){
    this.balance = balance;
}

public void checkBalance(){
    System.out.println("Current balance: " + this.getBalanceString());
}

public void makeDeposit(){
    Scanner input = new Scanner(System.in);

    System.out.println("Making a deposit...");
    System.out.println("How much do you want to deposit?: ");
    Double depositResult = input.nextDouble();
    this.balance += depositResult;
    }

    public void makeWithdrawal(){
        Scanner input = new Scanner(System.in);

        System.out.println("Making a withdrawal...");
        System.out.println("How much do you want to withdrawal?: ");
        Double withdrawalResult = input.nextDouble();
    }

}

abstract class User implements HasMenu, Serializable{

            String userName;
            String pin;

            public boolean login(){
                    boolean output = false;
                    Scanner input = new Scanner(System.in);
                    System.out.println("Please input username.");
                    String userNameResult = input.nextLine();
```

```
            if(userName.equals(userNameResult)){

                  System.out.println("Please input pin.");
                  String pinResult = input.nextLine();

                  if(pin.equals(pinResult)){
                        output = true;
                  }else{
                        System.out.println("pin is not correct");
                  }
            }else{
                  System.out.println("Username is not correct");
            }
            return login();
}

public boolean Login(String userName, String pin){
            return true;
}

public void setUserName(String userName){
                  this.userName = userName;
            }

      public String getUserName(){
                  return this.userName;
            }

            public void setPin(String pin){
                  this.pin = pin;
            }

      public String getPin(){
                  return this.pin;
            }

            public abstract String getReport();
}
```

3. Whenever I'm making a class, it should only have 1 reason to change, per the Single Responsibility Principle. Additionally, I should make any class I make small in that there is only one method and one line of code for each responsibility in the program. Plus, I should make sure that each of my classes has cohesion, which means I should have instance variables in each of my classes that can be manipulated by each of the methods in each of those classes.

Tab 5

Qaden Childs - Assignment 5

1. https://github.com/Dung30ns-Dr0g0ns546/Hello-World/blob/main/README.md
2. Holub emphasises that an object-oriented design shouldn't just hold data, but also encapsulate behavior and interact with one another in ways that mirror real-world responsibilities.
3. Before I read “An Extreme Programming Episode,” I mainly thought of Test-Driven Development (TDD) as just testing the code I’ve written before moving on to the main part, and Pair Programming (PP) as just two people sharing a keyboard to work on code at the same time. However, the article completely changed my mind on the matter for both. TDD is more than just going over code I’ve written to make sure it works, but instead is a step-by-step process where I write code that initially doesn’t work, before writing more where it does, then changing things around to improve it. As for Paired Programming, it's a paired collaboration where one person writes the code and the other thinks ahead. At the end of the day, this article has changed my perspective on these two topics.
4. After watching the video, I can easily say that I agree with most parts of it. For starters, not every programmer is perfect and is going to be successful on the first try. However, I was rather surprised to hear that not all code is language-dependent, meaning that not all code will be written in one language, like Java, C++, or Python. However, to say that I wasn’t also surprised by the age of the video and its topic would be a lie. However, to sum it all up, I do believe that the video is still valid to this day, just not on everything.

Tab 6

Qaden Childs - Assignment 6

1. After reviewing each of the F.I.R.S.T. principles, I can say that the most important of all of them would be Self-Validating. Anybody can have a test for a program run quickly, or not have them depend on each other, or make them repeatable. It’s only making sure that the test has a Boolean output, which should be the top priority. Whenever one sees an error, their first thought is to try and solve the issue to ensure that the program will work. However, the main focus shouldn’t be to make the code perfect, but only to make it so that it works.
2. https://github.com/bsu-cs222s1-fall25-smalviya/First_Project_Empty/blob/master/README.md