CADT IDT

C1-S5 PRACTICE

CLASSES & OBJECTS IN C++

At the end of this practice, you should be able to…

- ✓ **Implement** a class in C++ with specific **attributes** and **methods**
- ✓ **Instantiate objects** of the defined class and utilize methods to manipulate attributes.
- ✓ Examine the **object instantiation in memory** using debugger tools

How to compile your code?

Assuming your file is named: exercise.cpp:

- ✓ Open a **terminal** at your file location
- ✓ **Compile** your Program using the following command

```
g++ -o exercise exercise.cpp
```

- ✓ **Run** Your Program using the following command

```
./exercise
```

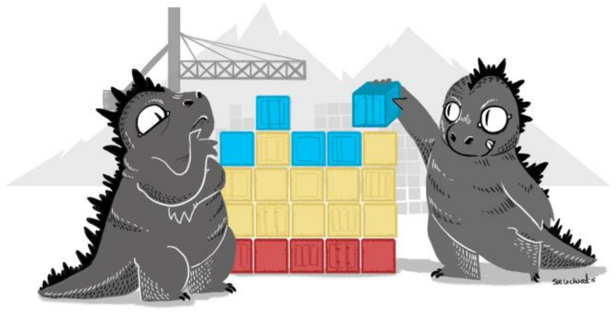
How to submit?

- ✓ Make a report PDF containing the screenshot of your program code and output for each exercise.
- ✓ **Submit your final source code report PDF to Microsoft team and turn it in.**

Are you lost?

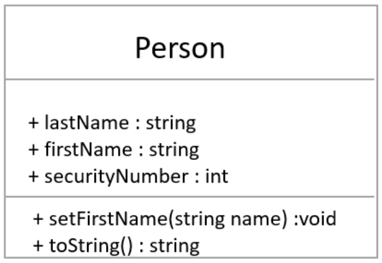
You can read the following documentation to be ready for this practice:

- ✓ Guide - WC3 School
- ✓ Guide – Class and Objects in C++
- ✓ Video – Class and Object in 10 minutes



EXERCISE 1

We want to create a data type to represent a person.
This person is identified by their last name, first name and social security number (integer).
We want to be able to enter and display this information.



Person class representation

Q1 - Write the Person class

- Add the needed attributes (public)
- Create the constructor with the needed parameters

Q2 – Instantiate the following 3 Persons in the main()

Last name	First name	Security Number
Ronan	Ogor	784-898

Q3 - What should be done to prevent users from manipulating the class attributes outside of the class?

- Make the appropriate changes

Q4 – Write a class method to update the first name of the person:

```
public setFirstName(string firstName) : void
```

Q5 – Write a method which return a string representation of the Person

```
public toString() : string
```

Q6 – Test you code

- Write the following test case:

```
Person ronan("ronan", "ogor", 4785);
ronan.setFirstName("ronano");
cout << ronan.toString() << endl;
```

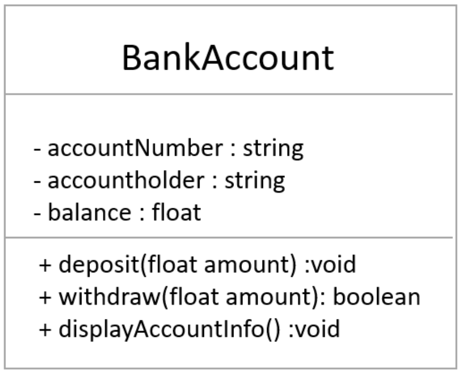
- Assess the output is:

```
First Name: ronano, Last Name: ogor, Security Number: 4785
```

EXERCISE 2

You have been hired as a software developer at a bank to help manage customer accounts!

Your task is to create a class that represents a bank account, allowing customers to deposit and withdraw money, while **ensuring that withdrawals do not exceed their balance.**


BankAccount
- accountNumber : string
- accountholder : string
- balance : float
+ deposit(float amount) :void
+ withdraw(float amount): boolean
+ displayAccountInfo() :void

BankAccount class representation

Q1 – Define a class named BankAccount

- Add private attributes: accountNumber (string), accountHolder (string), balance (float).

Q2 – Create the Constructor

- Create a constructor that initializes the accountNumber, accountHolder, and balance.

Q3 – Implement Member Functions:

```
public void deposit(float amount)
```

- Create a method deposit(float amount) that increases the balance by the specified amount.

```
public boolean withdraw(float amount)
```

- Make a method called withdraw(float amount) that takes money out of the balance, but only if there is enough money.
- If there isn't enough money, show a message saying there are not enough funds.
- This method gives back true if the operation worked, and false if it didn't.

```
public String toString()
```

- Create a method displayAccountInfo() that return a string representation of the account number, account holder's name, and current balance.

Q4 – Test you code

- Write the following test case:

```
// Create an account with 0$
BankAccount myAccount("ABC", "ronan", 0);
cout<< myAccount.toString() << endl;

// Deposite 100$
```

```
myAccount.deposit(100);
cout<< myAccount.toString() << endl;

// Withdraw 80$ - Should success
myAccount.withdraw(80);
cout<< myAccount.toString() << endl;

// Withdraw 30$ - Should fail
myAccount.withdraw(30);
cout<< myAccount.toString() << endl;
```

- Assess the output is:

```
Number: ABC, Holder: ronan, Balance: 0.000000
Number: ABC, Holder: ronan, Balance: 100.000000
Number: ABC, Holder: ronan, Balance: 20.000000
Number: ABC, Holder: ronan, Balance: 20.000000
```

EXERCISE 3

You are developing a graphics application that requires handling various geometric shapes!
One of the shapes you need to manage is a rectangle. You are given a Point2D class, and your task is to implement a Rectangle class that can calculate its perimeter and area and check if it is equal to another rectangle.

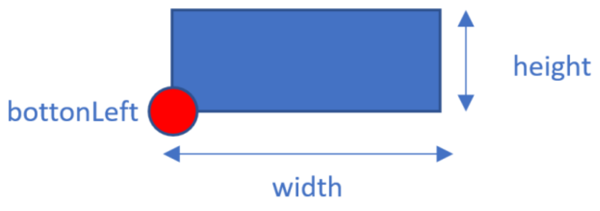
```
class Point2D {
public:
    double x;
    double y;

    Point2D(double x_val, double y_val) : x(x_val), y(y_val) {}

    double isEqual(const Point2D& other){
        return this->x==other.x && this->y==other.y;
     }
};
```

Q0 – Look at Point2D class

What does the method isEqual is doing?

Q1 – Define the **Rectangle** Class

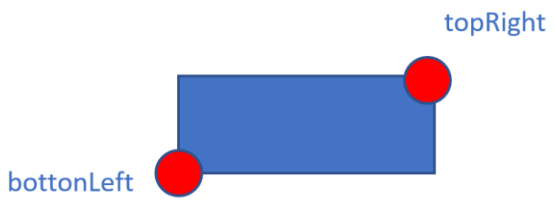
- Add **attributes**:
    - bottomLeft (Point2D)
    - width (double)
    - height (double).



Internally the Rectangle class keep 3 attributes…

Q2 – Create the constructor

- Create a constructor that takes two Point2D objects as parameters: bottomLeft and topRight.
- Inside the constructor, initialize the bottomLeft attribute and calculate the width and height from the given points.



…But to create a Rectangle, you need to pass 2 points

Q3 – Implement Member Functions:

Perimeter
Create a method `double perimeter()` that returns the perimeter of the rectangle.

Area
Create a method `double area()` that returns the area of the rectangle.

isEqual
Create a method bool `isEqual(const Rectangle& other)` that returns true if the current rectangle is equal to the other rectangle.

Q4 – Test your class

- Create many instances of Rectangle
- Compute and display their properties, and check for equality between rectangles

Q5 – Test you code

- Write the following test case:

```
Point2D p1(0,0);
Point2D p2(10,20);

Rectangle r1(p1, p2);
Rectangle r2(p1, p2);

string message =
     r1.isEqual(r2)? "rectangles are equal”: "rectangles are not equal";

cout<< message << std::endl;
```

- Assess the output is:

```
rectangles are equal
```

LET'S REFLECT ON OUR LEARNING!

After completing the exercise, reflect on the following questions to deepen your understanding and improve your problem-solving skills:

R1 - How well do you feel you understand the concept of classes and objects in C++ after completing these exercises?

R2 - Which specific aspects of this practice did you find most challenging, and why?