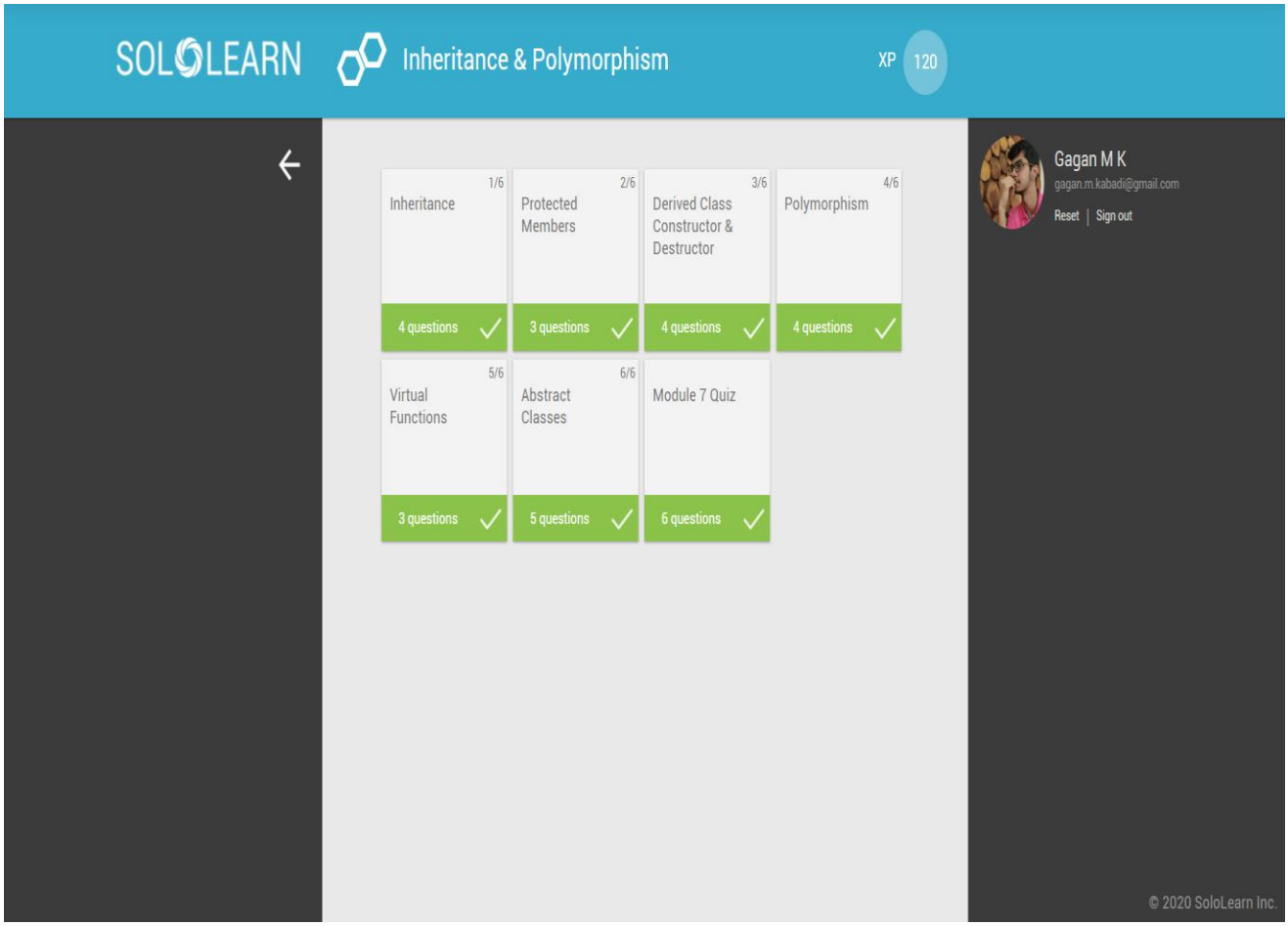
DAILY ASSESSMENT REPORT

Date:	**25 June 2020**	**Name:**	**Gagan M K**
Course:	**C Plus Plus**	**USN:**	**4AL17EC032**
Topic:	• **Inheritance & Polymorphism** • **Templates, Exceptions, and Files.**	**Semester & Section:**	**6th sem & 'A' sec**
GitHub Repository:	**Alvas-education-foundation/Gagan-Git**		

FORENOON SESSION DETAILS

Image of session



Report – Report can be typed or hand written for up to two pages.

Inheritance & Polymorphism:

- **Inheritance is one of the most important concepts of object-oriented programming.**
- **Inheritance allows us to define a class based on another class. This facilitates greater ease in creating and maintaining an application.**
- **The class whose properties are inherited by another class is called the Base class. The class which inherits the properties is called the Derived class. For example, the Daughter class (derived) can be inherited from the Mother class (base).**
- **A derived class inherits all base class methods with the following exceptions:**
  - Constructors, destructors
  - Overloaded operators
  - The friend functions
- **A class can be derived from multiple classes by specifying the base classes in a comma-separated list. For example: class Daughter: public Mother, public Father**
- **Up to this point, we have worked exclusively with public and private access specifiers.**
- **Public members may be accessed from anywhere outside of the class, while access to private members is limited to their class and friend functions.**
- **As we've seen previously, it's a good practice to use public methods to access private class variables.**
- **There is one more access specifier - protected.**
- **A protected member variable or function is very similar to a private member, with one difference - it can be accessed in the derived classes.**
- **Public Inheritance: public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. A base class's private members are never accessible directly from a derived class, but can be accessed through calls to the public and protected members of the base class.**
- **Protected Inheritance: public and protected members of the base class become protected members of the derived class.**
- **Private Inheritance: public and protected members of the base class become private members of the derived class.**
- **When inheriting classes, the base class' constructor and destructor are not inherited.**
- **However, they are being called when an object of the derived class is created or deleted.**
- **Constructors : The base class constructor is called first.**
- **Destructors : The derived class destructor is called first, and then the base class destructor gets called.**
- **This sequence makes it possible to specify initialization and de-initialization scenarios for your derived classes.**
- **The word polymorphism means "having many forms".**
- **Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.**

- **C++ polymorphism means that a call to a member function will cause a different implementation to be executed depending on the type of object that invokes the function.**
- **Simply, polymorphism means that a single function can have a number of different implementations.**
- **Polymorphism can be demonstrated more clearly using an example:**
- **Suppose you want to make a simple game, which includes different enemies: monsters, ninjas, etc. All enemies have one function in common: an attack function. However, they each attack in a different way. In this situation, polymorphism allows for calling the same attack function on different objects, but resulting in different behaviors.**
- **The previous example demonstrates the use of base class pointers to the derived classes. Why is that useful? Continuing on with our game example, we want every Enemy to have an attack() function.**
- **To be able to call the corresponding attack() function for each of the derived classes using Enemy pointers, we need to declare the base class function as virtual.**
- **Defining a virtual function in the base class, with a corresponding version in a derived class, allows polymorphism to use Enemy pointers to call the derived classes' functions.**
- **In some situations you'd want to include a virtual function in a base class so that it may be redefined in a derived class to suit the objects of that class, but that there is no meaningful definition you could give for the function in the base class.**
- **The virtual member functions without definition are known as pure virtual functions. They basically specify that the derived classes define that function on their own.**
- **A pure virtual function basically defines, that the derived classes will have that function defined on their own.**
- **Every derived class inheriting from a class with a pure virtual function must override that function.**
- **These classes are called abstract. They are classes that can only be used as base classes, and thus are allowed to have pure virtual functions.**
- **You might think that an abstract base class is useless, but it isn't. It can be used to create pointers and take advantage of all its polymorphic abilities.**

- **For example, you could write:**

```
Ninja n;
Monster m;
Enemy *e1 = &n;
Enemy *e2 = &m;

e1->attack();
e2->attack();
```

Templates, Exceptions, and Files:

- **Functions and classes help to make programs easier to write, safer, and more maintainable.**
- **However, while functions and classes do have all of those advantages, in certain cases they can also be somewhat limited by C++'s requirement that you specify types for all of your parameters.**
- **Function templates give us the ability to do that!**
- **With function templates, the basic idea is to avoid the necessity of specifying an exact type for each variable. Instead, C++ provides us with the capability of defining functions using placeholder types, called template type parameters.**
- **Template functions can save a lot of time, because they are written only once, and work with different types.**
- **Template functions reduce code maintenance, because duplicate code is reduced significantly.**
- **Enhanced safety is another advantage in using template functions, since it's not necessary to manually copy functions and change types.**
- **Function templates also make it possible to work with multiple generic data types. Define the data types using a comma-separated list.**
- **Let's create a function that compares arguments of varying data types (an int and a double), and prints the smaller one.**
  template <class T, class U>
- **T is short for Type, and is a widely used name for type parameters.**
- **It's not necessary to use T, however; you can declare your type parameters using any identifiers that work for you. The only terms you need to avoid are C++ keywords.**
- **Just as we can define function templates, we can also define class templates, allowing classes to have members that use template parameters as types.**
- **A specific syntax is required in case you define your member functions outside of your class - for example in a separate source file.**
- **You need to specify the generic type in angle brackets after the class name.**
- **To create objects of the template class for different types, specify the data type in angle brackets, as we did when defining the function outside of the class.**
- **In case of regular class templates, the way the class handles different data types is the same; the same code runs for all data types.**
- **Template specialization allows for the definition of a different implementation of a template when a specific type is passed as a template argument.**
- **Problems that occur during program execution are called exceptions.**
- **In C++ exceptions are responses to anomalies that arise while the program is running, such as an attempt to divide by zero.**
- **C++ exception handling is built upon three keywords: try, catch, and throw.**
- **throw is used to throw an exception when a problem shows up.**
- **A try block identifies a block of code that will activate specific exceptions. It's followed by one or more catch blocks. The catch keyword represents a block of code that executes when a particular exception is thrown.**

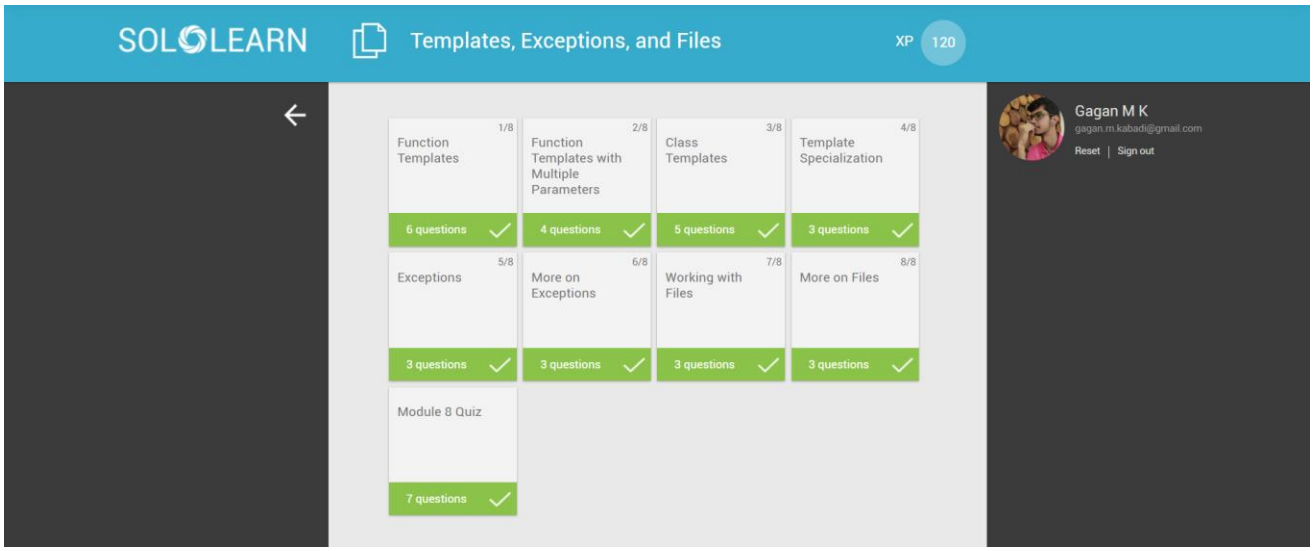
- **Code that could generate an exception is surrounded with the try/catch block.**
- **You can specify what type of exception you want to catch by the exception declaration that appears in parentheses following the keyword catch.**
- **Exception handling is particularly useful when dealing with user input.**
- **For example, for a program that requests user input of two numbers, and then outputs their division, be sure that you handle division by zero, in case your user enters 0 as the second number.**

```
int main() {
 int num1;
 cout <<"Enter the first number:";
 cin >> num1;

 int num2;
 cout <<"Enter the second number:";
 cin >> num2;

 cout <<"Result:"<<num1 / num2;
}
```

- **Another useful C++ feature is the ability to read and write to files. That requires the standard C++ library called fstream.**
- **Three new data types are defined in fstream:**
- **ofstream: Output file stream that creates and writes information to files.**
- **ifstream: Input file stream that reads information from files.**
- **fstream: General file stream, with both ofstream and ifstream capabilities that allow it to create, read, and write information to files.**



“Attended Webinar on VLSI Scope in INDIA conducted by Alva’s Education Foundation”