**1. Explain the benefits of object-oriented programming.**

Some of the benefits of OOP are modularity, ease of unit-testing, reusability, abstraction, delegation, and possible ease of event-driven programming. Modularity comes with the fact that each person in a group of ten can create one class with specific requirements, unit-test it, and combine it with the other classes. The specifics of others' classes are irrelevant and of no concern. Each can be tested individually, and then used within a bigger scope.

Unit-testing is easier when a logical grouping, such as a class, is formed. Each function can be listed and a more organized set of sections for each piece of functionality needing to be tested can be made. If that class is the only one in the file, it works well to have a unit test of a similar name to test the features of that class. Looking at the unit test would yield semi-rectangular blocks of text, with each block being one functionality.

Once written and tested, that's it, save updates. Classes can be written and tested once, and then reused elsewhere. This is the whole basis of every standard library, and all major third-party libraries. The code within is very often more efficient and robust than anything one could write themself. All one needs to do is reuse the good library instead of making a duplicated, sub-par one. Don't reinvent the wheel.

Classes often abstract aspects of one thing. For example, no one needs to know how to multiply two `BigInteger`s in order to do that with a class that abstracts that operation. Ideally, all one needs to do is `bigint1 \* bigint2`. Whether that causes repeated addition or optimized instructions is of no concern to the user in order to use it. Performance information is abstracted to a format such as Big-O notation. Multiplication is only one abstracted operation. When all such operations combine, they create a class that is much easier to use than the desired functionalities individually. Using a binary tree to store elements no longer requires knowledge of how to implement an (efficient) binary tree.

Classes interact with each other in ways that logically express relationships. Child classes can rely on parent class function implementations, or be forced to have one of their own. Once the classes have that relationship set up, they can be used together almost as if they are the same, relying on delegation from the parent to complete each task. Sprites can all move around differently, whether jumping up and down, or moving back and forth, or moving in the direction specified by input, but all sprites will often need to be moved at once. To do this, an array of sprites can be made, relying on the fact that each type of sprite will move in its own way. More formally known as polymorphism, the key is delegation from the base class to the proper derived class.

Lastly, classes can express relationships in the form of signals and slots. When one class meets a certain condition, it can send out a signal to all classes that have slots for that signal. It can pass the required information, and the other classes can each do what they will with it. In this way, it is very easy to see which classes are related to others in terms of events. For example, a Mouse class would emit MouseMove signals when the mouse is moved. Then, a class such as Recorder would record that movement for later playback. Other forms of the event-driven programming paradigm exist, but signals and slots are by far the best for classes.

**2. Discuss the difference between a class and an object.**

A class defines what each instance contains and can do, as well as what all instances share, and what the class can do without an instance. An object is an instance of the class that has its own copy of instanced data members, and can operate on them independently of other instances.

**3. Discuss the difference between properties and methods of a class.**

Properties of a class are stateful information associated with that class. Properties are useful for information that needs to persist between function calls. Methods can use properties, as well as any stateless information required to complete their task. Methods serve to accomplish something, and properties serve to hold information.

**4. What is code reuse? How does it relate to inheritance?**

Code reuse simply means that code can (and typically should) be reused instead of written from scratch. It relates to inheritance by the fact that base class function implementations can often be reused in derived classes. A derived class might simply extend the functionality of a base class, in which case all base class facilities would be reused.

**5. What is polymorphism? How does it relate to methods?**

Polymorphism relies on each derived class having a common function. With this assumed, that function can be invoked for each object of each class in a loop, rather than drawn out for each object. The concept is important for supporting plugins. Each plugin can implement specific functions that accomplish each possible task. The plugin could rely on the program’s natural behaviour, or override it, causing the actual effect of adding that plugin that the user sees.

For example, I made a notification area program with an option on the menu to add plugins. Each plugin was required to implement a specifically-named function that returned the structure of its submenu. The program’s main menu would list each submenu with the desired title, as well as all items on the submenu. The main program would handle selection of an item and call the appropriate function, as defined in the one required function. This assumes the plugin implements that function. Failure to do so would remove the item and issue a notification. Each plugin was stored in an array. Plugins themselves could be any DLL that exports said functions, whether created in C, C#, or VB. The point of the program was to group my many utilities in one local area, allowing each to offer the desired options. Although each was different, they could be grouped and used in a generic way due to that sort of polymorphism.

**6. Explain how the concepts of encapsulation and delegation relate to each other and how they differ** **from each other.**

Encapsulation provides a unified way of accessing properties. All access to the property is done through a function that can perform checks as it sees fit. If something needs to be added in the future, the interface doesn’t change, just the access function. No delegation is normally involved in direct relation to encapsulation. Their purposes are completely different. However, it is possible, depending on the language, to abuse delegation to subclass the class in question and override functions that are not meant to be overridden. In some circumstances, this has the potential to violate encapsulation.