**CSE 210: Programming with Classes**  
**W05 Assignment: Explain Inheritance**

**What is inheritance and why is it important?**

Inheritance is a fundamental principle of object-oriented programming that allows a class (called a *child* or *subclass*) to inherit the properties and behaviors (methods and attributes) of another class (called a *parent* or *base class*). This helps reduce code duplication and promotes code reuse, making it easier to maintain and extend programs.

One major benefit of inheritance is that it helps create a more organized and hierarchical structure in the code. For example, if several classes share common functionality, we can define that shared functionality in a base class and have the other classes inherit from it instead of rewriting the same code multiple times.

A practical example of inheritance is in a program where I created a base class called Activity, and then created specific activity types that inherited from it, such as Running, Swimming, and Cycling. Each subclass reused the common behavior defined in the Activity class and then added its own specific details.

Here’s a snippet of code from that project:

public class Activity  
{  
 protected DateTime \_date;  
 protected int \_duration;  
  
 public Activity(DateTime date, int duration)  
 {  
 \_date = date;  
 \_duration = duration;  
 }  
  
 public virtual string GetSummary()  
 {  
 return $"{\_date.ToShortDateString()} ({\_duration} min)";  
 }  
}  
  
public class Running : Activity  
{  
 private double \_distance;  
  
 public Running(DateTime date, int duration, double distance) : base(date, duration)  
 {  
 \_distance = distance;  
 }  
  
 public override string GetSummary()  
 {  
 return $"{base.GetSummary()} Running - Distance: {\_distance} km";  
 }  
}

In this example, the Running class inherits from the Activity class and reuses its constructor and GetSummary method. It also overrides the GetSummary method to add more specific details about running.

In summary, inheritance allows developers to write cleaner and more efficient code. It makes software easier to scale and update, especially when working with complex systems that involve multiple related types.