Inheritance

Inheritance allows a new class to be based on existing class. It inherits all member variables and functions (except constructors and destructors) from the base class.

Inheritance can be thought of implementing the idea of "is a" relationships.

> A smartphone is a computer -> smartphone inherits from a computer.
>
> A dog is an animal -> dog inherits from animal.
>
> A student is a person -> student inherits from person.

"is a" doesn't always mean there is inheritance.

> Julie is a student -> doesn't not mean Julie inherits from student. Instead, Julie is an instance of a student.
>
> Prof. Chuck is a jerk -> Instead this means I have property with a value indicating I'm a jerk

How do we know? Inheritance involves taking an object and adding new details to that object (member variables and functions).

> Smartphone adds a phone number variable to a computer. It adds the call function to a computer.
>
> A dog adds an owner variable and trick function to an animal.

To inherit in c++, Say:

> class derivedClass: accessSpecifier baseClass{…};

Example:

> class Dog: public Animal{…}

The access specifier determines how the members of the base class are treated in the derived class. If left off, it defaults to private.

		Access Specifier		
		public	private	protected
Base Class Members	Public	Public	private	Protected
	Private	X	X	X
	Protected	protected	Private	protected

Checkpoint

15.4 – protected can be used in inherited classes. Private can't.

15.5 – class access is used with inheritance. Member is used inside and outside of a class.

15.6 – Private

- a - inaccessable
- b - private
- c - private
- SetA - private
- SetB - private
- SetC - private

Protected

- a - inaccessable
- b - protected
- c - protected
- SetA - protected
- SetB - protected
- SetC – protected

Public

- a - inaccessable
- b - protected
- c - protected
- SetA - protected
- SetB - public
- SetC – public

Private is the default.

Protected members are like private members to instance but like public members to class members and derived class members.

Constructors and Destructors with base classes
Constructors are called in order from lowest base to highest derived class.
Destructors are called in order from highest derived to lowest base class.

You can pass arguments to the base class constructors.

- className::className(ParameterList) : BaseClassName(ArgList){…}

If the base class doesn't have a default constructor, you need to call it. If it does have a default construction, you don't need to call it.

- className::className(ParameterList){…} if the base class has a default constructor.

Checkpoint
15.7 –

- Entering Sky
- Entering Ground
- Leaving Ground
- Leaving Sky

15.8 -

Entering Sky
Entering Ground
Leaving Ground
Leaving Sky