6-11-2024
WEDNESDAY

OOP 4301

Faisal Hussein sir

* In Java, even the "main" function is defined inside a class.
* A code is always expected to be refactored. However, adding new codes is preferred to modifying older ones.
* ~~Br~~ Behaviours should be added where it belongs (depending on the requirement).
* A Pointer is a special type of variable that holds addresses.
* A memory space is reserved just for the program's variables.
* * ⇒ Deference/Indirection Operator
* All types of pointers occupy the same size. However, mismatched assignment isn't allowed (except for void pointers).

HW ✱ const int* ≠ int* const

✱ "new" is an operator which replaces DMA malloc in C++.

example: int* i = new int [size]; → If omitted, a single variable/object is created.

✱ DMA is a runtime operation.

✱ "delete" is used to deallocate DMA.

example: delete[] arr;

for arrays

If not used ~~property~~ the error of Runaway Pointer may occur while space is wasted.

✱ Creating a pointer doesn't trigger the constructor.

✱ A base class' pointer can point to its derived classes' objects. Vice-versa isn't allowed. However, the former type of pointer can only call/access the base members (functions). Also, it won't call the functions overriden in the derived classes.

* When a pointer is declared, only its type or class definition is used.
* To make the base class' pointer access the derived class' members, the function to be overriden should be virtual. In that case, the base class' pointer will check its contents (i.e., class type that's assigned) and look for overriden functions first. It'll return to the base class' definition only if no overriden function is found.

7-11-2024
THURSDAY

OOP4301
Faisal Hussein sir

* C# adds some advantages over Java (such as: embedded SQL), but it must work with a SQL server.
* For assigning an address to a pointer, the compiler checks the compatibility between the types.
* A Virtual Function is a function whose behaviour can be overridden within an inheriting class with a function with the same signature. It allows the pointers late/dynamic binding as it checks the type of the object pointed at by the base class' pointer & call the overridden function if it's present.
* In C++, the keyword "override" isn't mandatory. "@override" is necessary to be mentioned before the method for Java. Such conventions ~~check for~~ prevent errors.

* base_ptr = &drv_object; can't call members not present in the base class (error: "class [base]' has no member named '---'").
* Virtual Functions ⇒ 1 interface, multiple methods
  ⇒ example of polymorphism
* Late/dynamic binding allows runtime polymorphism.
* Virtual functions are hierarchial.
* Base class' pointers support the address of objects from ~~multiple~~ classes which used multi-level inheritance.
* Because of the property of inheritance, derived objects and pointers can access members (which are accessible) defined in the base class.
* Pointers only stores one address. In case of contiguous memory spaces, it's the 1st cell.
* Error message for drv_ptr = &base_object;

"Invalid convertion from *[base] to *[derived]."

* Lack of overridden countempart will lead the virtual function back to the base class.

* "override" keyword in the derived class won't compensate for the lack of "virtual" keyword in the base class.

* An inherited class must be careful about the base class' constructors. If it's 0 arg., it can be omitted; otherwise the initializer list must be used.

* Even for multi-level inheritance, for a virtual function, the object's overridden function will be called, otherwise the compiler will return to the base class and the intermediate classes will be ignored (however, the properties of inheritance make this principle obsolete).

* base_ptr = &dm_obj; allows single interface for multiple methods.
* Virtual functions are used for run-time executions.
* A blank func. for a base class may allow its virtual nature to ensure elegant & dynamic func. calling for its derived classes.
* A Pure Virtual Function —
  - is blank (only the prototype is mentioned, & not the body)
  - derived classes must override this func.
  - isn't really there, but we assume its virtual existence to ensure derived classes' functionalities
* virtual type funcName() = 0; ⇒ Pure Virtual Function
* If a PVF is not overriden in a derived class, a compilation error will pop up.
* If a class has PVF, it's an Abstract Class. It cannot have objects and cannot be instantiated

~~det~~ directly. If the derived class doesn't override the PVF, it also becomes an abstract class. However they can have pointers.

* An object of an abstract class can't be created as it isn't fully defined. Pointers are possible due to polymorphism.

* instantiation ⇒ creating an object