Inheritance

Object relationships

Inheritance

- Inheritance is one of the four main OOP concepts
  - Encapsulation / Data Protection
  - Inheritance
  - Polymorphism
  - Abstraction

Inheritance

- Inheritance is a modelled as an ‘is-a’ relationship
  - Class B, though distinct, is descended from Class A; it is a form of class A (only differs in certain aspects)
- Inheritance allows classes to
  - Share what is common between them
    - Data and Methods
  - Modify only that which is different

Inheritance

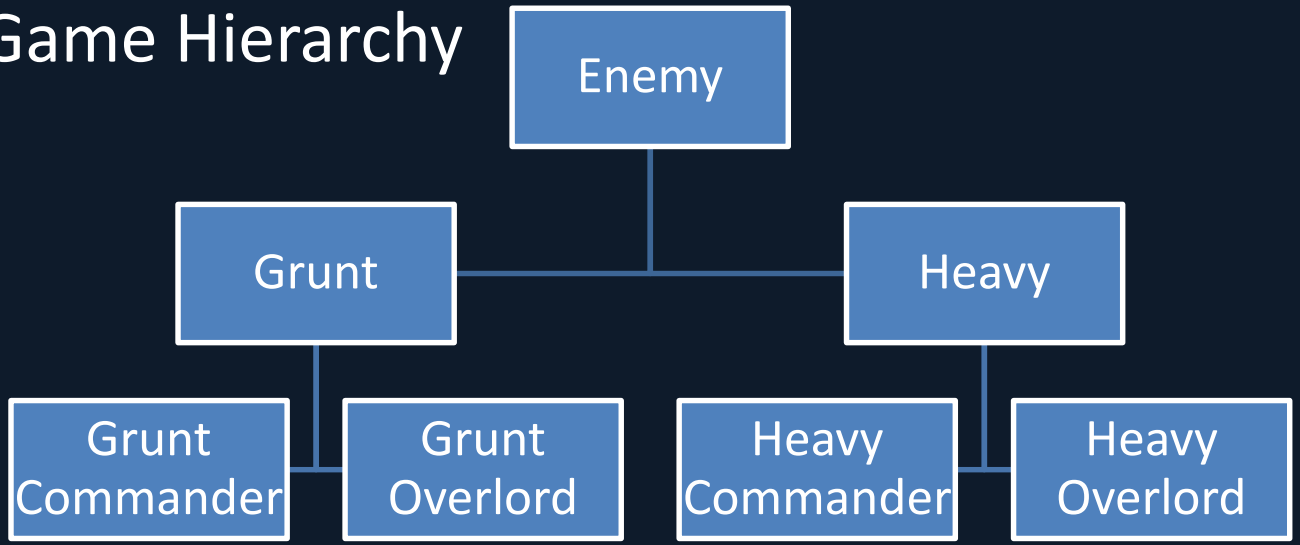
- Definitions
  - Base Class
    - A class that is not derived from any other class
  - Derived Class
    - A class derived from a base class. It has either modified functionality or data compared to the base class

Inheritance

- If Object A is derived from object B we say that Object A inherits properties from Object B
  - In some systems, the inheritance tree can become quite deep
  - Good idea to keep it as shallow as possible
    - To avoid confusion
    - To reduce the complexities of abstract nouns

Example

- Game Hierarchy



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Code Example

- Inheritance syntax is straightforward
  - Derived class declaration begins with its name followed by the base class name

```
class Position2D
{
public:
    float x, y;
};

class Enemy
{
public:
    Enemy();
    void Update();

protected:
    Position2D position;
};
```

```
//Grunt can access Enemy::position as it
now
//derives from grunt
class Grunt : public Enemy
{
public:
    void DoGruntStuff();

private:
    int id;
};
```

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Protected Keyword

- In-between public and private.
  - Protected members are inaccessible to outside classes, just like private
  - Protected members are inherited
  - Private members are not
  - Therefore protected members can be accessed in derived classes.

Constructors

- What about constructors for the base class?
  - The enemy class now has a default and a non-default constructor

```
class Enemy
{
public:
    Enemy();
    Enemy(Position2D &pos)
    {
        position = pos;
    }
    void Update();

protected:
    Position2D position;
};
```

Constructors

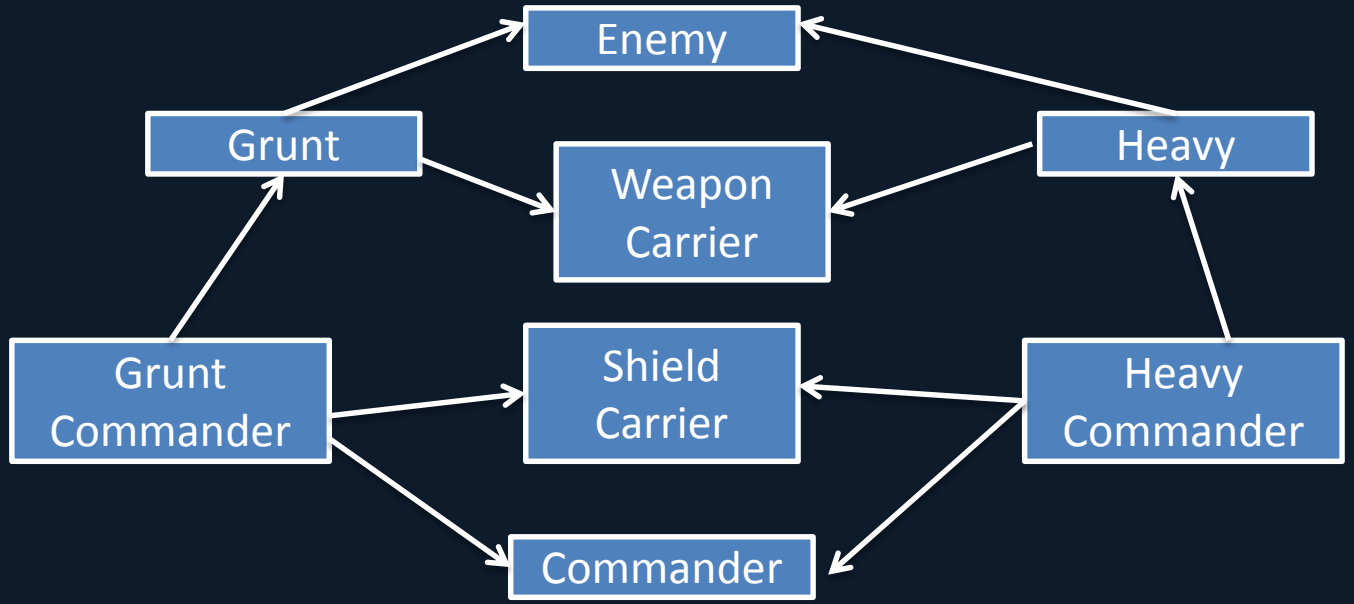
- Grunt now has default and non-default constructors
  - The default automatically calls Enemy's constructor first
  - The non-default explicitly calls Enemy's constructor

```
class Enemy
{
public:
    Enemy();
    Enemy(Position2D &pos)
    {
        position = pos;
    }
    void Update();

protected:
    Position2D position;
};
```

```
class Grunt : public Enemy
{
public:
    //Calls Default Enemy Constructor
    Grunt()
    {
        id = 0;
    }

    //Explicity calls enemy constructor
    Grunt(Position2D position, int id) : Enemy(position)
    {
        this->id = id;
    }

    void DoGruntStuff();

private:
    int id;
};
```

Multiple Inheritance

- Derived classes may have multiple base classes



Multiple Inheritance

- Can lead to issues
  - Especially a diamond inheritance tree
  - Are the base classes shared or independent?
  - Do the grunt commander or heavy commander share base classes or keep unique copies?

Multiple Inheritance

- In general, multiple inheritance complicates software and should be avoided
- There may be times when it is needed, if required, use sparingly

Summary

- Inheritance is an extremely powerful and useful OO concept.
- It allows us to define relationships between classes.
- Use sparingly, as too many classes can make the code confusing to follow.
- Use the protected keyword to retain privacy, but allow derived classes to access still.