Chess Program 7 - Inheritance Part A, extending Piece as a Rook.

1. Now we are going to inherit from `Piece`, creating a `Rook` class that extends `Piece`.
2. Add `rook.cpp` and `rook.hpp` files. Copying `piece.cpp` and `piece.hpp` file is a reasonable starting point.
3. Declaring that `Rook` will extend `Piece` is done in the class header. For now the `Rook` constructor looks much like the `Piece` constructor. Our `Rook` class will override the `getSymbolAt()` method. The entire `Rook` class definition will be fairly short, and look like so,

```
#include "piece.hpp"
class Rook : public Piece
{
public:
    Rook(int row, int column, char type);       // constructor
    char getSymbolAt(int row, int column);      // override
};
```

4. The `Rook` constructor in `rook.cpp` must call the `Piece` constructor in its definition, passing any parameters along as needed, e.g.

```
Rook::Rook(int row, int column, char type) : Piece(row, column, type)
{
    cout << "Rook object being created. " << this << endl;
}
```

5. Your rook `getSymbolAt()` implementation should behave like the implementation in `Piece`, but only need take into account the case the `Piece` being of type `ROOK`, e.g.

```
char Rook::getSymbolAt(int indexRow, int indexColumn)
{
    if (indexColumn == mColumn && indexRow == mRow)
    {
        return mType;
    }
    else if (indexRow == mRow || indexColumn == mColumn)
    {
        return 'X';
    }
    else
    {
        return ' ';
    }
}
```

6. Note: you will need to change your `Piece` member variables from `private` to `protected` so that the rook's `getSymbolAt()` method has access to them.
7. For now, lets simply `new` up our extending `Rook` class instead of `Piece` in `Chess` and see what happens. Change `Piece* mPtrPiece;` in the Chess header to `Rook* mPtrPiece;`, `#include "rook.hpp"` becomes `#include "piece.hpp"`, and in the body method `makeChessPiece()` use `new Rook(...` instead of `new Piece(...`
8. Now all your pieces will all draw as rooks, and we know the Rook class is working.