Classes in C++

similar to classes in Java
template

```
	class className
	{
		public:
			...

		private:
			...
	};
```

a class is just a definition (like a struct)
	therefore, it does not allocate memory
	and again, it is considered a long statement and thus requires a semicolon at the end

so what's all this public/private stuff?
	these are called access modifiers
	public implies that whatever is placed there can be accessed by anything inside/outside of the class
	private implies that whatever is placed there can only be accessed by members inside the class
	order of public/private is unimportant

e.g.

```
	class clockType
	{
		private:
			int hr;
			int min;
			int sec;

		public:
			void setTime(int, int, int);
			void getTime(int&, int&, int&) const;
			void printTime() const;
			void incrementSeconds();
			void incrementMinutes();
			void incrementHours();
			bool equalTime(const clockType&) const;
	};
```

clockType facts:
- 7 member functions (public)
- 3 member variables (private; cannot be accessed outside the class)
- all functions can access the variables (no need for passing them as parameters)
- equalTime takes a clockType as parameter (by reference)
  - this is usually how we'll pass classes because it's more efficient
  - why?
  - and why the const?

const after a function prototype indicates that the function cannot modify the member variables

how do we use classes in our programs?

e.g.

```
clockType myClock, yourClock;


myClock.setTime(10, 30, 0);
myClock.printTime();
yourClock.setTime(x, y, z);
if (myClock.equalsTime(yourClock))
```

illegal:

```
myClock.hr = 10;                    // hr is private
myClock.min = yourClock.min;        // again, min is private
```

built-in operations in classes

- dot operator (.)
  - to access members (variables and functions)
- assignment (=)
  - to assign one class to another; e.g.
    - myClock = yourClock;
  - so we can perform some aggregate operations on classes
  - this copies all variable members from yourClock to myClock
  - the result is two objects that have the same hr, min, sec (two different copies)
  - so they are copied (by value) → we call this a member-wise copy

arrays of classes

```
clockType myClocks[2];

myClocks[0].printTime();
```

classes can be passed to functions and returned from functions

```
void addOneHour(clockType c)
{
        c.incrementHours();
}
```

- but remember that c is passed by value, so the hours will not be changed when we return
- how to fix this so the change is permanent?
  - pass the clockType by reference!
    - void addOneHour(clockType &c)

efficiency

- sometimes classes can be huge
- passing by value can take time
- so we would like to pass it by reference
- but this allows the function to change the object which is bad
- we solve this by allowing a class to be passed by reference and by also specifying it to be constant
  - void addOneHour(const clockType &c)

- so now c is passed by reference (efficient)
- and c cannot be changed within the function (const)
- the following statements are illegal within this function:
  - c.setTime(1, 2, 3);
  - c = otherClock; // (assuming otherClock is defined previously in this function)

getters/setters/constructors in C++

- e.g.

```
int getHr() const
{
        return hr;
}

void setHr(int hours)
{
        hr = hours;
}
```

- constructor
  - C++ does not automatically initialize variables (including private variables)
  - sometimes (usually) we may want to do this
  - we use a constructor
    - has the same name as the class and initializes variables
  - default constructor has no parameters
  - properties
    - name of constructor is the same as the class
    - is a function but has no type (neither void nor value-returning)
    - a class can have more than one constructor (they all have the same name)
      - must have different formal parameters
    - executed automatically when an object of the class type is declared
    - which constructor executes depends on the types of values passed to the object

HANDOUT classes1

HANDOUT classes2