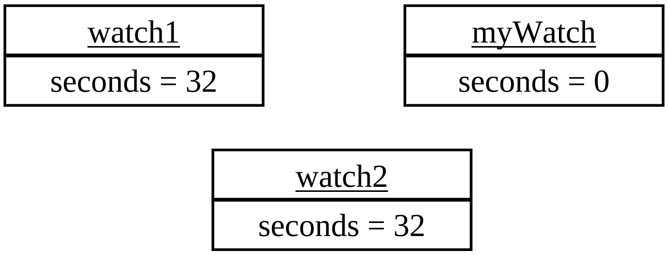
4. *Objects:Identity, State & Behavior*

Object Identity

Distinguishes object by their inherent existence & not by descriptive properties that they may have.



watch1

seconds = 32

myWatch

seconds = 0

watch2

seconds = 32

Identity - an Handle to the Object

C++ - Memory Address is an Object Identifier

“this” pointer

Each object has a variable called “this”. “this” is a pointer. It holds the address (Identity) of the Object.

watch1.this “is equal to” &watch1

watch2.this “is equal to” &watch2

string1.this “is equal to” &string1

“this” helps “self-reference” & to pass “self” to other objects.

Behavior & State of an Object

- Methods take an Object from one State to Another
- A method may be called only when an Object is in a selected set of states.
  - Example: FileHandler:
    - Open may be called only if state is not open
    - Close may be called only if the state is open
- Conditions: Pre-Conditions & Post-Conditions
  - Pre-Condition (Advertised Requirements)
    - Must be satisfied for proper/guaranteed execution of function.
  - Post-Condition (Advertised Promises)
    - Guaranteed State of the Object upon completion of function

Behavior & State of an Object...

Example:

```
class Stack {
    ...
    push(Item& objC);
        // Requirement: Stack not full.
        // Promise: size = size +1; pop() == objC.

    Item* pop();
        // Requirement: Stack not empty
        // Promise: size = size - 1
};
```

Some OOPLs like Eiffel Support pre/post Conditions

No Direct C++ Support!

- Specified through Comments
- Enforced through Exception Handling

const functions

- Within a const function - no modification to object members allowed
- What if you want to change a member (that does not really represent state of an object)
  - Example: keeping track of number of reads to an object

```
class Record {...
   int readCount; …
   String getRecordId() const
   {…
     readCount = readCount + 1; // Error. Not allowed
   }
};
```

castaway and mutable

- casting away the pointers - bad practise

```
class Record {...
     int readCount; …
     String getRecordId() const  {…
       ((Record*)(this))->readCount = readCount + 1;
                        //Getting a non const pointer from this
     }
};
```

- mutable key word - safe and portable

```
class Record {...
     mutable int readCount; …
     String getRecordId() const
     {…
       readCount = readCount + 1; // OK since readCount is mutable
     }
};
```

Class Members & Methods

Common to & Shared by All Objects.

Class Members (Variables)

- Represents a concept based on the abstraction
- Shared by all Objects of a Class

Class Methods (functions)

- Works on the general concept rather than specific Object
- May be based on the class Members

Example of a Static Member

Count of Number of Objects of a Class

```
class Bacteria {
    static unsigned long count;
    ...
public:
    Bacteria() { count = count + 1; ... }
    ~Bacteria() { count = count - 1; ... }
    ...
};
```

```
unsigned long Bacteria::count = 0;
```

Example of a Static Method

A method in class Bacteria ...

```
static unsigned long getCount() { return count; }
```

Usage:

```
    Bacteria b1;
    b1.getCount();              // Will return 1
    Bacteria b2;
    b1.getCount();              // Will return 2
    b2.getCount();              // Will return 2
```

Static Method called on Objects.

```
    Bacteria::getCount();      // Will return 2
```

Another Example of Static Method

```
class DBMgr {
    static DBMgr* themgr;
    DBMgr() { }              // No way to create a DBMgr outside of this Class!!
public:
    static DBMgr* getDBMgr()      // Only way to create a DBMgr. Controlled.
    {
            if (themgr == 0)
                    themgr = new DBMgr;
            return themgr;
    }
};
DBMgr* DBMgr::themgr = 0;
Usage:
    DBMgr* dbmgrptr = DBMgr::getDBMgr();        // Created if one does not exist.
```

Singleton Pattern

Modules and Namespaces

- Large project has several modules of code
- Modularizing the system makes it more understandable and maintainable
- In UML modules are called Components
- C++ implements packages using namespaces

Namespaces in C++

```
namespace Accessories {
    class Wheel {}; // belongs to the Accessories
    class Mirror{};  // belongs to the Accessories
};
namespace CarModule {
    class Engine {}; // belongs to the CarModule
    class Mirror{};// belongs to the CarModule
    class Car {
            Engine* pEngine;   // No scope resolution needed
            Accessories::Wheel* pWheel[4]; // Need resolution
            Mirror* pRearView; // Mirror that belongs to CarModule
            Accessories::Mirror* pSideMirror[2]; // Mirror belongs to Accessories
    public:             …
            void drive();
    };
};
void CarModule::Car::drive()
{// drive function's code
}
```

namespaces : mechanism for logical grouping. Has scope

Using Declaration

- Convenience to avoid redundant resolution
- Local synonym for entity in another namespace

```
void maintainCar(CarModule::Car& car)
{
    using CarModule::Engine;

    Engine& theEngine = car.getEngine();
            //Engine is a synonym for CarModule::Engine
    …
    CarModule::Mirror& theMirror= car.getRearViewMirror();
}
```

Using Directives

- Namespace directives may be used for convenience

```
void maintainCar(CarModule::Car& car)
{
    using namespace CarModule;

    Engine& theEngine = car.getEngine();
          //Engine is a synonym for CarModule::Engine

    //...

    Mirror& theMirror= car.getRearViewMirror();
}
```

Namespace Clashing

- Two or more namespaces have same class, function, etc.

```
void maintainCar(CarModule::Car& car)
{
    using namespace Accessories;
    using namespace CarModule;
    Engine& theEngine = car.getEngine();
    //...
    Mirror& theMirror= car.getRearViewMirror();
}
```

error C2872: 'Mirror' : ambiguous symbol

- Use explicit resolution

```
void maintainCar(CarModule::Car& car)
{
    using namespace Accessories;
    using namespace CarModule;
    Engine& theEngine = car.getEngine();
    //...
    CarModule::Mirror& theMirror= car.getRearViewMirror();
}
```

Lab Work: Details provided on-line.