Classes, compilers, dependencies

```cpp
#include <string>
#include "day.h"

typedef string TimeRange;

class Appointment
{
public:
    TimeRange duration();
    void print(ostream & output);

private:
    Day myDay;
};
```

- why use `ostream` instead of `#include <stream.h>`
- what is a typedef and how is it used?
- make depend for Appointment/ostream?
- changes to `Day` force recompile for appointment clients?

.h guidelines, preprocessor in action

```cpp
#include <string>
#include "day.h"

typedef string TimeRange;

class Appointment
{
public:
    TimeRange duration();
    void print(ostream & output);

private:
    Day myDay;
};
```

- minimize `#includes` in every `.h` file
  - avoid circular dependencies
  - avoid re-compile by minimizing dependencies
- class Foo in foo.h, class Bar in bar.h, client foobar.cpp

```cpp
#ifndef _FOO_H
#ifndef _BAR_H
#define _FOO_H
#define _BAR_H
#include "bar.h"
#include "foo.h"

class Foo
{
    Bar getBar();
    Foo getFoo();
};
```

- Avoid `#includes`, use forward references, sometimes you must do this as shown above (even if you don’t want to)

#include “foo.h”

- will be needed in `.cpp` file, e.g., `foo.cpp` and `bar.cpp`
- using pointers and references in `.h` files minimizes dependencies
  - minimize recompiles when `.h` changes
  - loose coupling; avoid implementation dependencies when possible
- avoid letting implementation leak into public view
  - what about private section?
  - opaque pointer: `FooImpl * myImpl;`
    - implementation of `FooImpl` is hidden, class can be implemented in `foo.cpp` (handle-body idiom)
  - factory: inheritance hierarchy, ABC

Software frameworks/cadavers

- Experience with OO programming and design shows that design patterns are useful
  - Where do we get the experience?
  - How do we impart experience?
  - What can we use to illustrate patterns in practice?
  - What patterns should we emphasize, how, when?

  *Good design comes from experience, experience comes from bad design*
  
  Fred Brooks/Henry Petroski

- Good design also comes from experiences with good design
  - use software cadavers to show patterns in practice
Factory Methods/Factory Pattern

- Suppose we’re using a Printer hierarchy in KWIC (e.g., print key word in CAPS, left justified lines, etc.) Where does the Printer object being used come from?
  - The real program needs a Printer object, but doesn’t know what kind it needs, just when it’s needed
  - Separate creation/construction from use, localize knowledge of which (or how) a Printer is created

- Create a factory method aka virtual constructor
  - We can change the factory code, or create new factories

  Printer * p = factory.makePrinter();
  // use p here, don’t know what kind we have

Factories continued

- Suppose you want to experiment/use different implementations as part of KWIC
  - Use hashmap, treemap, trie, super-duper-wingdoodle, ...
  - The interfaces for classes doesn’t need to change, e.g., consider BSTMap and HMap in Tapestry, both implement tmap (similar in Java, as we’ll see)
  - We may have more than the Map-like class, there may be associated Printers, Iterators, ...

- Abstract Factory defines how to create products used by clients: Map, Printer, Iterator, FileReader, ...
  - Clients create a concrete factory and use it, hashfactory, triefactory, ...
  - Consider Motif, Mac, Windows, ..., LAF for GUIs

C++ idioms

- What happens with the statement myDay = d; ?
  - assignment is memberwise unless operator = overloaded
  - copy constructor used in passing parameters by value
- If you need one of: destructor, assignment operator, copy constructor, you need all of them
  - heuristic only: managing resources other than memory
  - preventing objects from being copied
  - what about non-copyable state, e.g., stream

- In assignment operator, watch for self-assignment
- Study implementation of string/vector

copy constructor

- Used for “first-time” creation
  ```cpp
  template <class Item>
  tvector(const tvector<Item> & vec)
  // precondition: Item supports assignment
  // postcondition: return copy of vec
  {
    // allocate storage
    myList = new Item[myLength=vec.myLength];
    assert(myList != 0);
    // copy elements
    for(int k = 0; k < vec.myLength; k++)
    {
      myList[k] = vec.myList[k];
    }
  }
```

- Used for pass-by-value
  ```cpp
  DoStuff(Date d);
  //...
  Date first(1,1,2000);
  DoStuff(first);
  ```

- what about use of myLength in code as opposed to length()?