Two main goals:

 Encapsulate knowledge about which concrete classes the system uses

2. Hide how instances of these classes are created and built

- System at large knows about objects through their interfaces defined by abstract classes
- Give us flexibility in:
 - what gets created
 - who creates it
 - how it gets created
 - when it gets created

- Singleton
- Factory Method
- Abstract Factory
- Builder
- Prototype



Creational Patterns: Singleton

- Consider a class called Logger
 - Logs information to a file
 - Needed by many different parts of an application

Creational Patterns: Singleton

Logger.h

return court

C++ Programming

Const Correctness, Part 1

Recall ...

- const is a keyword declaring a type constraint that indicates that certain data cannot be modified
- Note that this does not imply that the data is read-only in memory
- The constraint is enforced by the compiler

```
const int answer = 42;
answer = 43; // compilation error!
```

const and Pointers

• Both pointers and the data they point to can be const

Use the right-to-left rule to read

```
const(int** a)
int* const(* b))
const int* const * c
const int* const * const d
```

```
// pointer to a pointer to a const int
const int** a
int* const * b
const int* const * c
const int* const * const d
```

```
// pointer to a pointer to a const int
const int** a
// pointer to a const pointer to an int
int* const * b
const int* const * c
const int* const * const d
```

```
// pointer to a pointer to a const int
const int** a
// pointer to a const pointer to an int
int* const * b
// pointer to a const pointer to a const int
const int* const * c
const int* const * const d
```

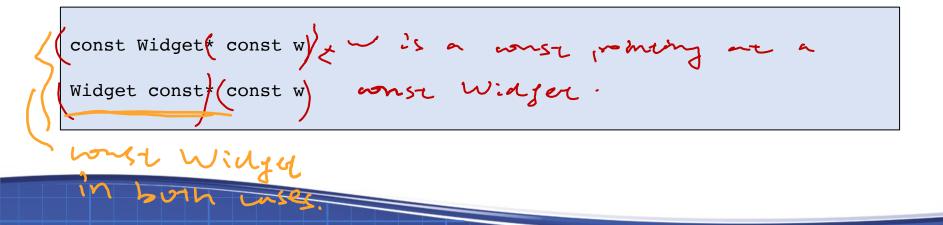
```
// pointer to a pointer to a const int
const int** a
// pointer to a const pointer to an int
int* const * b
// pointer to a const pointer to a const int
const int* const * c
// const pointer to a const pointer to a const int
const int* const * const d)
```

```
is the data inself vonst?
is the primer vonst?
```

```
char* (const h) = "Hello"; h' is a pointer to worst h
char* w = "World";
h[1] = 'u'; data in h is nor// will it compile?
h = w; priver iself 25 worse. // will it compile?
                              No.
pointing he pointing to man.
```

const and Pointers

- When the data pointed to is constant, some add const before the type name; others add it after
- You will see both in the real world



Pointer Assignments Involving const

Address of non-const object can be assigned to a const pointer

```
int i = 4;

(const-int(*)) = &i;

s's a pointe eo onse ine dara.
```

• In this case, we are promising not to change an object that was previously okay to change

Pointer Assignments Involving const

You cannot assign the address of const object to a non-const pointer

```
const int i = 4; wast int *j // this maches the type.

int* j = &i) // compilation error! So it works here.

j prancing at wast int.
```

• The second line causes a compilation error because we're saying that we might change i through the pointer

Pointer Assignments Involving const

Exception: string literals

```
char* c = "Hello";
```

- "Hello" is a const char *, but we can assign it to a char *
- Explanation from the horse's mouth:

This is allowed because in previous definitions of C and C++, the type of a string literal was char*. Allowing the assignment of a string literal to a char* ensures that millions of lines of C and C++ remain valid. It is, however, an error to try to modify a string literal through such a pointer.

- Bjarne Stroustrup, The C++ Programming Language