Exceptions

When Good Code Goes Bad

Analysis

- Object-orientation is all about recognizing the different "actors" at work in the system being modeled.
  - First, the different data available within the system are organized appropriately.
  - Secondly, functionalities relating to the state of data and its management are bound to that data.

Analysis

- Object-orientation is all about recognizing the different "actors" at work in the system being modeled.
  - These objects ("actors") may then interact with other objects through well-formed, bounded relationships.

Analysis

1. Constraints: should our object have limitations imposed on it, beyond those implied by the language we're using?
   - Some of our internal state variables (fields) may allow values which make no sense in the context of what our object represents.

Encapsulation

- In a previous lecture, we built a "Person" class with well-defined states and behavior.
  - This involved setting up accessor and mutator methods to ensure the object held a logical, valid state.
  - What we *didn't* cover was how an object should behave when something tries to make its state invalid.

A First Object

```
public class Person
{
    private:
        const string name;
        int age;

    public:
        Person(string name, int age)
        string getName();
        int getAge();
        void haveABirthday();
}
```

A First Object

```
string Person::getName()
{
    return this->name;
}

int Person::getAge()
{
    return this->age;
}
```

A First Object

```
public void haveABirthday()
{
    this->age++;
}
```

Errors

- What if someone initializes our class incorrectly? (Bad "input"?)
  - Naïve solution: be "passive-aggressive."
  - To be more technical, we *could* refuse to change our object for bad inputs.
  - The problem here is that this is **impossible** for constructors to do. They *must* return an instance.
  - Going passive-aggressive would result in an invalid instance of our object! Not cool!

Errors

- A further problem with going "passive-aggressive" is that this gives *no* indication to outside code that something actually *has* gone wrong.
  - It's left up to outside code to realize that something is amiss… and if this is ignored, it causes strange behaviors later that can get tricky to test for.

Errors

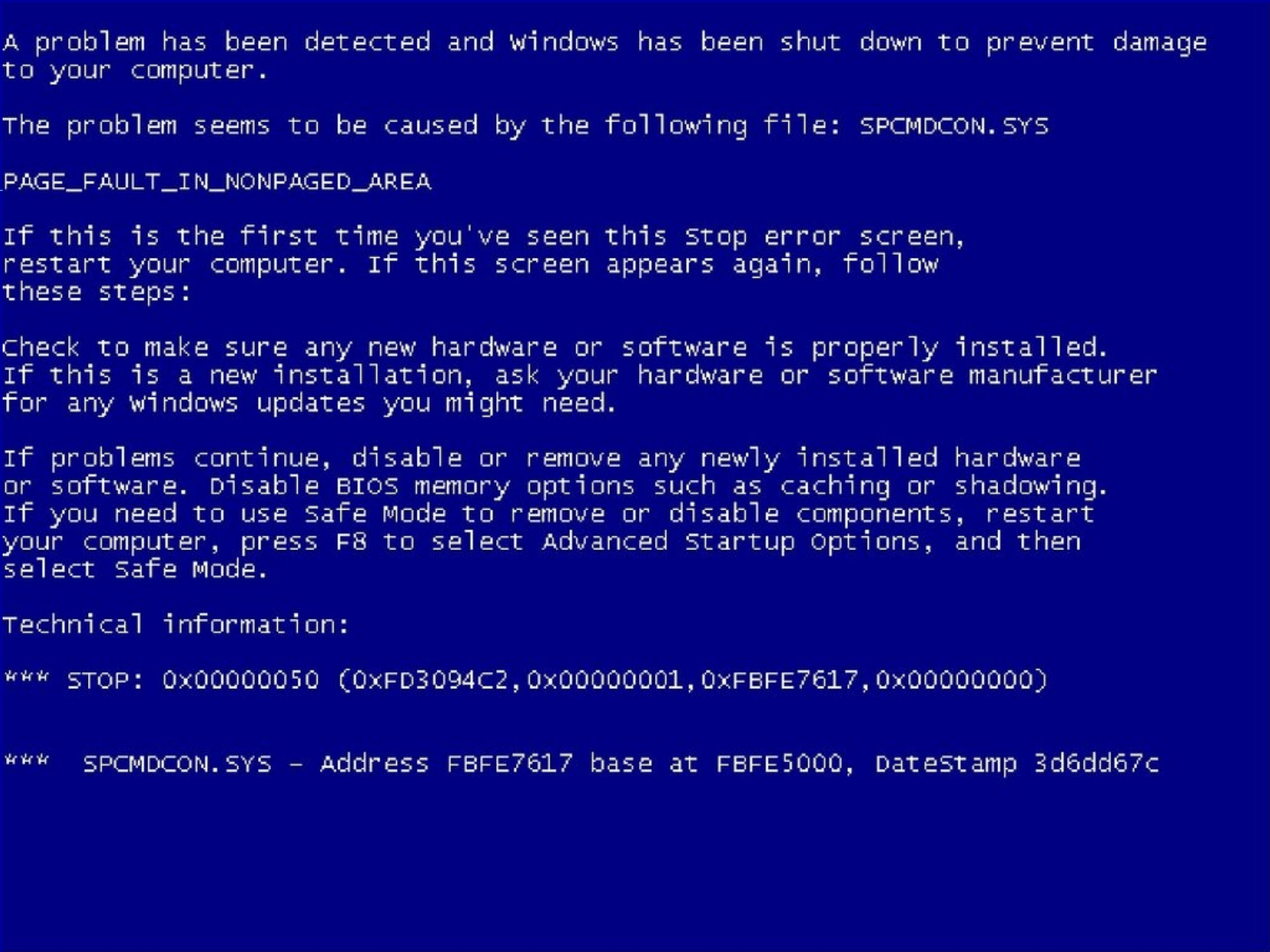
- What we'd really like to do: we'd like to indicate – forcibly, if necessary – that some sort of error has occurred within the program.
  - This "error" may have happened due to different reasons.
  - Could be the result of a mistake by a programmer…
  - Or it could be the result of a mistake by the program's user.

Errors

- Before examining how we can signal that an error has occurred, let's examine other sources of errors.
  - For example, dividing an integer by zero is one classic source of an error.
  - In calculator programs, this could easily be a user's actual request.

Errors

- What usually happens whenever a program has an error?


A problem has been detected and Windows has been shut down to prevent damage
to your computer.
The problem seems to be caused by the following file: SPCMDCON.SYS
PAGE_FAULT_IN_NONPAGED_AREA
If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:
Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.
If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.
Technical information:
*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)
*** SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c

Error Handling

- Many programming languages have built-in support for “catching” errors that occur.
  - This allows programs to attempt a graceful recovery…
  - Or to crash to the desktop safely instead.

Error Handling

- One mechanism used for signaling an error is that of the *exception*.
  - The *exception* operates by interrupting the standard flow of a program by jumping to specialized code for correcting the error.
  - Note that once an error is encountered within a line of code, it is not safe to evaluate the lines after it, as they may be affected by the error.

Error Handling

- In hardware, this is accomplished through something actually known as an *interrupt*.
  - Interrupts cause the CPU to cease evaluation of a program temporarily to handle something time-sensitive.
  - There are mechanisms in place to resume operation properly, however.

Error Handling in C++

- Within C++, many such *exceptions* are represented by a form of the std::exception class.
  - Use #include <stdexcept>.
  - In particular, most object-oriented exceptions are represented this way.
  - Other sorts of errors (e.g. arithmetic) may not be represented this way.
  - Note – these are "object" exceptions.