C++ class design 1

Richel Bilderbeek

October 13, 2015

Classes

Programs must be written for people to read, and only incidentally for machines to execute

H. Abelson & G.J. Sussman

It's hard to overstate the value of simple design and clear code

Sutter & Alexandrescu 2005

```
//Every organism has a fitness and trait
std::vector<double> fitnesses;
std::vector<double> traits;
assert(fitnesses.size() == traits.size());
```

P.1

```
class organism {
  //
};

std::vector<organism> organisms;
```

- P.1: Express ideas directly in code

```
enum class sex { male, female };

///mammals cannot change sex during their life
class mammal {
  sex m_sex;
};
```

P.1

```
enum class sex { male, female };

class mammal {
  const sex m_sex;
};
```

- P.1: Express ideas directly in code
- Drawback: class will not by copyable anymore, need to define the big five

```
void draw(int x1, int y1, int x2, int y2);
```

C.1

```
void draw(point from, point to);
void draw(point from, size sz);
```

- C.1: Organize related data into structures (structs or classes)
- If data is related, that fact should be reflected in code.
- Note: A simple class without virtual functions implies no space or time overhead.

- What would the rule be?

```
struct Coordinat { //struct here
  double x;
  double y;
  std::string label;
};

class Date { //class here
public:
  Date(Year y, Month m, Day d);
private:
  Year m_y;
  Month m_m;
  Day m_d;
};
```

- Use class if the class has an invariant; use struct if the data members can vary independently

```
class player {
public:
  void kill() { m_health = 0; }
private:
  int m_health;
};
```

F.6

```
class player {
public:
  void kill() noexcept { m_health = 0; }
private:
  int m_health;
};
```

- F.6: If your function may not throw, declare it noexcept

```
class player {
public:
  int get_health() noexcept;
private:
  int m_health;
};
```

C.6

```
class player {
public:
  int get_health() const noexcept;
private:
  int m_health;
};
```

- C.6: Declare a member function that does not modify the state of its object const
- More precise statement of design intent
- Better readability
- More errors caught by the compiler
- (More optimization opportunities)

```
class tic_tac_toe {
public:
  const std::array<int,9>& get() const noexcept {
    return m_squares;
  }
private:
  std::array<int,9> m_squares;
};
```

C.3

```
class tic_tac_toe {
public:
  square get(const int x, const int y) const noexcept;
private:
  //Do not care how this is implemented
};
```

- C.3: Represent the distinction between an interface and an implementation using a class
- Allow the user not to know what kind of data types you used in the private section of your class [1]
- improves readability
- simplifies maintenance
- [1] Bjarne Stroustrup. The C++ Programming Language (4th edition). 2013. ISBN: 978-0-321-56384-2. Chapter 16.4. Advice. page 479

```
class my_class {
public:
  int get() const noexcept;
  std::string to_str() const {
    return std::itos(get());
  }
private:
  int m_x;
};
```

C.4

```
class my_class {
public:
  int get() const noexcept;
private:
  int m_x;
};

std::string to_str(const my_class& c) {
  return std::itos(c.get());
}
```

- C.4: Make a function a member only if it needs direct access to the representation of a class

- Which to prefer?

```
class point1 {
  int x, y;
  //  ... operations ...
  // .. no virtual functions ...
};

class point2 {
  int x, y;
  // ... operations, some virtual ...
  virtual ~point2();
};
```

C.10

```
class point1 {
  int x, y;
  //  ... operations ...
  // .. no virtual functions ...
};
```

- C.10 Prefer a concrete type over more complicated classes

```
class person {
public:
  //
private:
  int m_id; //All ID's are unique
}

int main() {
  const person a;
  const person b = a;
  assert(a != b);
}
```

C.11

```
const my_class a;
const my_class b = a;
assert(a == b);
```

- C.11: Make concrete types regular

Next

- class design 2: big five

Legal stuff

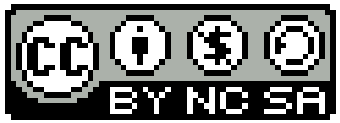

Figure 1: CC-BY-NC-SA

Download at:

`www.github.com/richelbilderbeek/CppPresentations/class_design1.pdf`

Figure 2: GitHub

Send feedback by adding an issue or doing a pull request.