Outline
- Classes, objects, instances
- Different kinds of constructors
- Const objects
- Const data member
- Const member functions
- Enumerated types
- Examples

Class Terminology
- Class is a data type.
- An object is an instance of a class.
- Class interface is the outside view of class instances.
- Class interface consists of all public members (data and functions).

Constructors
- A class may have several constructors.
  - Constructors must have different signatures (list of arguments).
  - Default constructor
    - Either no arguments or all arguments have default values but not both
  - Copy constructor
    - Used when objects are initialized

Example
```cpp
class Beer {
    char *name;
    int taste;
    public:
        Beer(); // default constructor
        Beer(char *name, int taste=2);
        Beer(const Beer &); // copy constructor
        char *get_name() { return name }
        int get_taste() { return taste; }
        ~Beer();
};
```

Constant Objects
- Constant object cannot be changed after initialization.
- Static declaration:
  ```cpp```
  ```cpp```
- What is the taste of bud?
- Allocating const object with new:
  ```cpp```
  ```cpp```
Constant Data Members

- The value of a constant data member is assigned at initialization and cannot be changed.
- Prepend member declaration with `const`.

```cpp
class Beer {
    const char *name;
    int taste;
public:
    const char *get_name() const { return name; }
}
```

Constant Member Functions

- Constant member function do not changed any object data members.
- General form:

```cpp
Type function_name(...) const;
```
- Constant member function do not changed any object data members.

```cpp
class Beer {
public:
    const char *get_name() const { return name; }
    int get_taste() const { return taste; }
}
```

Mix and Match

- Const function can only call other const functions.
- Only const functions may be called on const objects.
- Const object may be passed only to functions with const argument.
- Any function may be called on non-const objects.

Enumerated Types

- General form:

```cpp
enum tag {name1 = val1, name2 = val2,...}
```
- Tag and val1, val2 are optional.
- Values must be integers.
- Example:

```cpp
enum BeerTaste {bad, soso, ok=5, good, mmm=10}
```

```cpp
class Beer {
    enum BeerTaste taste;
}
```