

Bjarne Stroustrup from *The C++ Programming Language*, 3rd Edition, page 223:

The aim of the C++ class construct is to provide the programmer with a tool for creating new types that can be used as conveniently as the built-in types.

A type is a concrete representation of a concept.

For example, the C++ built-in type `double` with its operations `+`, `-`, `*`, etc., provides a concrete approximation of the mathematical concept of a real number. A class is a user-defined type.

We design a new type to provide a definition of a concept that has no counterpart among the built-in types.

A program that provides types that closely match the concepts of the application tend to be easier to understand and to modify than a program that does not.

A well-chosen set of user-defined types makes a program more concise. In addition, it makes many sorts of code analysis feasible. In particular, it enables the compiler to detect illegal uses of objects that would otherwise remain undetected until the program is thoroughly tested.

The C++ class type provides a means to encapsulate heterogeneous data elements and the operations that can be performed on them in a single entity.

Like the `struct` type, a `class` type may contain data elements, called data members, of any simple or structured type (including class types).

Also like the `struct` type[†], a `class` type may also contain functions, called function members or methods, that may be invoked to perform operations on the data members.

The class type also provides mechanisms for controlling access to members, both data and function, via the use of the keywords `public`, `private` and `protected`.

A variable of a class type is referred to as an object, or as an instance of the class.

[†] What's the difference? Members of a struct type are, by default, public; members of a class type are, by default, private.

```
class USD {
private:
    int mAmount;           // store value as total cents; signed

public:
    USD();                 // constructors, initialize declared variables
    USD(int Amount);

    void Round();         // operations on USD variables
    void roundUp();
    void roundDown();

    . . .
};
```

```
USD::USD() {  
    mAmount = 0;  
}  
  
USD::USD(int Amount) {  
    mAmount = Amount;  
}
```

```
int main() {  
  
    USD A(125),      // represents USD 1.25  
        B(153);    // represents USD 1.53  
    USD C;         // represents USD 0.00  
    . . .  
}
```

```
int main() {  
  
    USD A(125),      // represents USD 1.25  
        B(153);    // represents USD 1.53  
    USD C;          // represents USD 0.00  
    . . .  
    A.Round();     // represents USD 1.00  
    B.Round();     // represents USD 2.00  
}
```

```
void USD::Round() {  
  
    int Dollars = mAmount / CENTSPERUSD;  
    int Cents   = mAmount % CENTSPERUSD;  
  
    if ( Cents > 50 ) {  
        mAmount = (Dollars + 1) * CENTSPERUSD;  
    }  
    else {  
        mAmount = Dollars * CENTSPERUSD;  
    }  
}
```

```
class USD {
private:
    int mAmount;           // store value as total cents; signed

public:
    USD();                 // constructors, initialize declared variables
    USD(int Amount);

    . . .

    bool isDebit() const;
    std::string toString() const;

    . . .
};
```

```
string USD::toString() const {

    ostringstream Out;
    Out << mAmount / CENTSPERUSD
        << SEPARATOR
        << setfill('0') << setw(2) << mAmount % CENTSPERUSD
        << setfill(' ');

    return Out.str();
}
```

```
class USD {
private:
    int mAmount;           // store value as total cents; signed

public:
    USD();                 // constructors, initialize declared variables
    USD(int Amount);

    . . .
    USD operator+(USD RHS) const;
    USD operator-(USD RHS) const;
    . . .
};
```

```
USD USD::operator+(USD RHS) const {
    return USD(mAmount + RHS.mAmount);
}

USD USD::operator-(USD RHS) const {
    return USD(mAmount - RHS.mAmount);
}
```

```
int main() {  
  
    USD A(125),      // represents USD 1.25  
        B(153);     // represents USD 1.53  
    USD C;          // represents USD 0.00  
    . . .  
    C = A + B;  
    cout << "C: " << C.toString() << endl;  
    . . .  
}
```

```
class USD {
private:
    int mAmount;           // store value as total cents; signed

public:
    USD();                 // constructors, initialize declared variables
    USD(int Amount);
    . . .
    bool operator==(USD RHS) const;
    bool operator!=(USD RHS) const;
    bool operator<=(USD RHS) const;
    bool operator<(USD RHS) const;
    bool operator>=(USD RHS) const;
    bool operator>(USD RHS) const;
    . . .
};
```

```
int main() {  
  
    USD A(125),      // represents USD 1.25  
        B(153);     // represents USD 1.53  
    USD C;          // represents USD 0.00  
    . . .  
    C = A + B;  
    cout << "C: " << C.toString() << endl;  
  
    C = C - B;  
    if ( C == A ) {  
        cout << "Subtraction worked correctly." << endl;  
    }  
    . . .  
}
```