Operator Overloading

Chapter 14

Operators in C++

```
+    -    *    /    %    ^    &    |    ~
!    =
<    >    +=   -=   *=   /=   %=   ^=
&=   |=   <<   >>   >>=  <<=
==   !=   <=   >=   &&   ||   ++   --
->*  ,    ->   []   ()   new  delete,
sizeof
```

```
int X = 1, Y = 3;
Z[1] = X + Y;
if (X == 3 && Y > 0) {

}
```

An **overloaded operator** *redefines* the behavior of an operator with respect to a class object

```
string s1("I love ");
string s2("overloaded operators!");
string s3 = s1 + s2;
cout << s3[0] << endl;

vector<int> v1(10);
v1[4] = 42;
```

The operators <<, +, and [ ] have been overloaded!

The overloaded operator + is implemented with a special method defined in the string class!

```
string s1("I love ");
string s2("overloaded operators!");
string s3 = s1 + s2;
cout << s3 << endl;
```

When C++ sees this line of code, it makes a call to the method operator+()

```
string s1("I love ");
string s2("overloaded operators!");
string s3 = s1.operator+(s2);
cout << s3 << endl;
```

The overloaded operator [] is implemented with a special method defined in the vector class!

```
vector<int> v1;
v1.push_back(1);
cout << v1[0] << endl;
```

When C++ sees this line of code, it makes a call to the method operator[]()

```
vector<int> v1;
v1.push_back(1);
cout << v1.operator[](0) << endl;
```

You can overload any C++ operator except the following:

```
?:      .      ::      .*
```

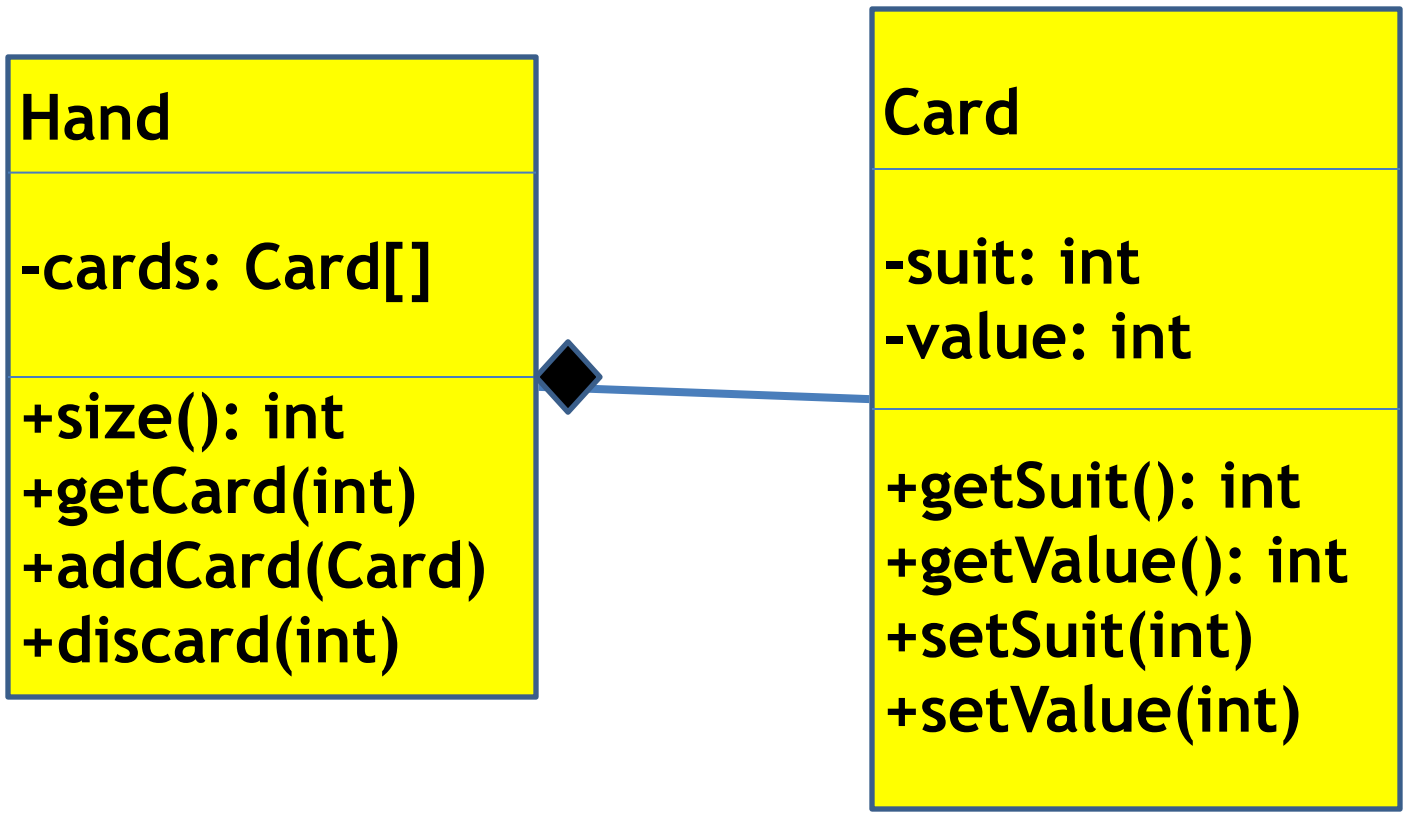
E.g.

```
int X = ( Y > 1 ? 0 : 1);
cout << p.getName();
cout << City::population;
```

Running example: Cards and Card Hands

- We are going to be making a **Card** class, with properties **suit** (int) and **value** (int)
  - We are going to re-define the behavior of '**<<**' and '**==**' for the **Card** class
- We are going to make a **Hand** class, to model a player's hand in a card game
  - We are going to re-define the behavior of '**<<**', '**+**', and '**[]**' for the Hand class

UML


Hand
-cards: Card[]
+size(): int
+getCard(int)
+addCard(Card)
+discard(int)
Card
-suit: int
-value: int
+getSuit(): int
+getValue(): int
+setSuit(int)
+setValue(int)

Initial Card Class

```
class Card {
private:
   int suit;
   int value;

public:
   static const int SPADES = 1;
   static const int DIAMONDS = 2;
   static const int CLUBS = 3;
   static const int HEARTS = 4;
   Card(int s, int v) {
      setSuit(s);
      setValue(v);
   }

   int getSuit() { return suit; }
   void setSuit(int s) {
      suit = (s >= SPADES && s <= HEARTS) ? s : SPADES;
   }
   int getValue() { return value; }
   void setValue(int v) {
      value = (v >= 1 && v <= 13) ? v : 1;
   }
};
```

We want to be able to output a card object

```
void showCard() {
   if (value == 13) cout << "King";
   else if (value == 12) cout << "Queen";
   else if (value == 11) cout << "Jack";
   else if (value == 1) cout << "Ace";
   else cout << value;
   cout << " of ";

   switch (suit) {
   case SPADES:
      cout << "Spades";
      break;
   case DIAMONDS:
      cout << "Diamonds";
      break;
   case CLUBS:
      cout << "Clubs";
      break;
   case HEARTS:
      cout << "Hearts";
      break;
   }
}
```

This all works, but it's kinda weak

Overload operators

showCard works OK:

```
cout << mycard.showCard() << endl;
```

But it's ugly

C++ uses << for string objects, can we use it for Card objects?

Overloaded operators!

Overload output operator for Card

```
friend ostream& operator<<(ostream& str, const Card& c) {
   if (c.value == 13) str << "King";
   else if (c.value == 12) str << "Queen";
   else if (c.value == 11) str << "Jack";
   else if (c.value == 1) str << "Ace";
   else str << c.value;
   str << " of ";

   switch (c.suit) {
   case SPADES:
      str << "Spades";
      break;
   case DIAMONDS:
      str << "Diamonds";
      break;
   case CLUBS:
      str << "Clubs";
      break;
   case HEARTS:
      str << "Hearts";
      break;
   }

   return str;
}
```

```
vector<Card*> hand;
for (int c = 0; c < hand.size(); c++)
   cout << *(hand[c]) << endl;
```

A **friend class** in C++ can access the private and protected members of the class in which it is declared as a friend.

Compare Cards

I also want to compare to see if two cards are equal

```
bool Card::operator==(Card& c) {
    return (getSuit() == c.getSuit() &&
            getValue() == c.getValue());
}
```

```
Card aceDiamonds(1, 2);
Card aceSpades(1, 1);
Card aceDiamonds2(1, 2);

cout << aceDiamonds == aceSpades << endl;
cout << aceDiamonds == aceDiamonds2 << endl;
```

We can overload for a Hand of cards, too!

`+`	to add a card to the hand
`+=`	to support the shortcut
`<<`	to output a hand
`[]`	to get a specific card

Example

```
class PlayerHand {
private:
   vector<Card*> cards;

public:
   int size() { return cards.size(); }

   PlayerHand& operator+(Card* c) {
      cards.push_back(c);
      return *this;
   }

   PlayerHand& operator+=(Card* c) {
      return *this + c;
   }

   Card* operator[](int i) {
      return cards[i];
   }

   friend ostream& operator<<(ostream& str, PlayerHand& hand) {
      for (int c = 0; c < hand.size(); c++)
         str << (*hand[c]) << endl;
      return str;
   }
};
```

Full example

You can get the entire code base from https://github.com/ptucker/CardGame/