CS116

Today

- What are exceptions?
- Example: Geometric Objects
- Introduction to Inheritance
- Examples:
  - Geometric Objects
  - *Queue* and *Stack* starting from *List*

What are Exception

- Run-time anomalies

- Throwing Exceptions

- The Try Block

- Catching an Exception
  - Exception Objects
  - Stack Unwinding
  - Rethrow
  - The Catch-All Handler

- Exception Specification
-

Example - Throw

```
int fib (int N)
{
   if (N < 0)
      throw N;
   ...
}
```

Example - Try

```
int main ()
{
   try {
      fib (-10);
   } catch (int e) {

   }
   return 0;
}
```

Catch

- Particular Exceptions
- Any Exception
- Mechanism: Stack Unwinding

re-throw

```
int
main ()
{

 try {
  fib (-10);
 } catch (int e) {
  throw;
 }
 return 0;
}
```

Specification

```
int
fib (int N) throw(int)
{
  if (N < 0)
    throw N;

  if (N ==0)
    throw string("break");

  ...
}
```

Object Composition

- Composition by value
  - other objects are declared members

- Composition by reference
  - only references or pointer are stored to other objects

Geometric Objects

- Point
- Line
- Device

Point

```
class Point {
  double X, Y;
public:
  Point ();
  Point (double, double);
  ~Point ();
  double getX () const;
  double getY () const;
  double setX (double);
  double setY (double);
  void draw (ostream&) const;
  void draw (odevice&) const;
  friend ostream& operator << (ostream&, const Point&);
  friend odevice& operator << (odevice&, const Point&);
};
```

Line

```
class Line {
  Point X, Y;
public:
  Line ();
  Line (Point&, Point&);
  ~Line ();
  Point getX () const;
  Point getY () const;
  Point setX (Point);
  Point setY (Point);
  void draw (ostream&) const;
  void draw (odevice&) const;
  friend ostream& operator << (ostream&, const Line&);
  friend odevice& operator << (odevice&, const Line&);
};
```

Device

```
class odevice {
  ostream *out;

public:
  odevice (ostream* out);

  ostream& getContext () const;

  friend odevice& operator << (odevice&, int);
  friend odevice& operator << (odevice&, double);
  friend odevice& operator << (odevice&, string);

};
```

Test

```
#include "point.hh"
#include "line.hh"
using namespace geometric_objects;
#include "odevice.hh"
using namespace output_devices;

int main (int argc, char *argv[])
{
  Point p1 (1, 2), p2 (2, 1);
  Line l1 (p1, p2);

  cout << p1 << " + " << p2 << " => " << l1 << endl;
  odevice o (&cout);
  o << p1 << " + " << p2 << "\n\t => " << l1 << "\n";
  return 0;
}
```

Inheritance

- Code re-use
- Relations between classes (sub-types)

Object Oriented Design

List

--> Queue

--> Stack

Base Class

```
class List {
 ...

};
```

Derived Classes

```
class QueueException : public ListException {
 public:
  QueueException (string s) : ListException (s) { }
};
```

Derived Classes

```
class Queue : protected List {
 public:
  Queue () : List () { }
  Queue (Queue& q) throw (ListException) : List (q) { }

  void Enqueue (int) throw (ListException);
  void Dequeue (int&) throw (ListException);
  void Empty ();
  bool isEmpty ();
  Queue& operator = (const Queue& l) throw (ListException);
  friend std::ostream& operator << (std::ostream& out, Queue& q);

};
```

Base Class Member Access

```
void  Queue::Enqueue (int i) throw (ListException)
{
 List::Insert (i, 1);
}

void Queue::Dequeue (int& i) throw (ListException)
{
 if (List::isEmpty ())
   throw QueueException ("queue is empty");
 else
   List::Delete (i, List::Length ());
}
```

Derived Class Constructor

```
Queue () : List () { }
Queue (Queue& q) throw (ListException) : List (q) { }
```

Derived Class Destructor

Constructor Order

```
class A {
public:
  A () { cout << " create an object of type A\n"; }

};

class B : A{
public:
  B () : A () { cout << " create an object of type B\n"; }

};
```

Destructor Order

```
class A {
public:
   ~A () { cout << " destroy an object of type A\n"; }

};

class B : A{
public:
   ~B () { cout << " destroy an object of type B\n"; }

};
```

Public, Private& Protected

- public -> type inheritance
- private -> implementation inheritance
- protected -> ?

Inheritance&Composition

```
class Stack : protected List {
 public:
  Stack () : List () { }
  Stack (Stack& q) throw (ListException) : List (q) { }

  void Push (int) throw (ListException);
  void Pop (int&) throw (ListException);
  void Top (int&) throw (ListException);
  void Empty () { List::Empty (); }
  bool isEmpty () { return List::isEmpty (); }

  Stack& operator = (const Stack&) throw (ListException);
  friend std::ostream& operator << (std::ostream& out, Stack&
q);
};
```

Object Composition

- Composition by value
  - other objects are declared members

- Composition by reference
  - only references or pointer are stored to other objects

Inheritance&Composition

```
class Stack {
  List l;
 public:
  Stack () { }
  Stack (Stack& q) throw (ListException) { l = (List&)q; }

  void Push (int) throw (ListException);
  void Pop (int&) throw (ListException);
  void Top (int&) throw (ListException);
  void Empty () { l.Empty (); }
  bool isEmpty () { return l.isEmpty (); }

  Stack& operator = (const Stack&) throw (ListException);
  friend std::ostream& operator << (std::ostream& out, Stack& q);
};
```

Inheritance&Composition

```
class Stack {
  List *l;
 public:
  Stack () { l = new List (); }
  Stack (Stack& q) throw (ListException) { l = new List ((List&)q); }

  void Push (int) throw (ListException);
  void Pop (int&) throw (ListException);
  void Top (int&) throw (ListException);
  void Empty () { l->Empty (); }
  bool isEmpty () { return l->isEmpty (); }

  Stack& operator = (const Stack&) throw (ListException);
  friend std::ostream& operator << (std::ostream& out, Stack& q);
};
```

Next Time

- **Inheritance (again *Chapter 17*)**
- **Polymorphism**