Exceptions

Announcement

- Today is the last lecture for the course.
- Next week, we will have an in-lab challenge as practice for End-sem Exam.
  - Please go directly to the lab at 2 PM.

Introduction

- A systematic, object-oriented approach to handling errors.
- Errors are exceptional circumstances which can only be detected at runtime.
  - Running out of memory while allocating using `new`
  - Opening a file which does not exist
  - Accessing a vector at out-of-bounds index
  - …
  - User-defined classes can specify their own exceptions.

Why Exceptions?

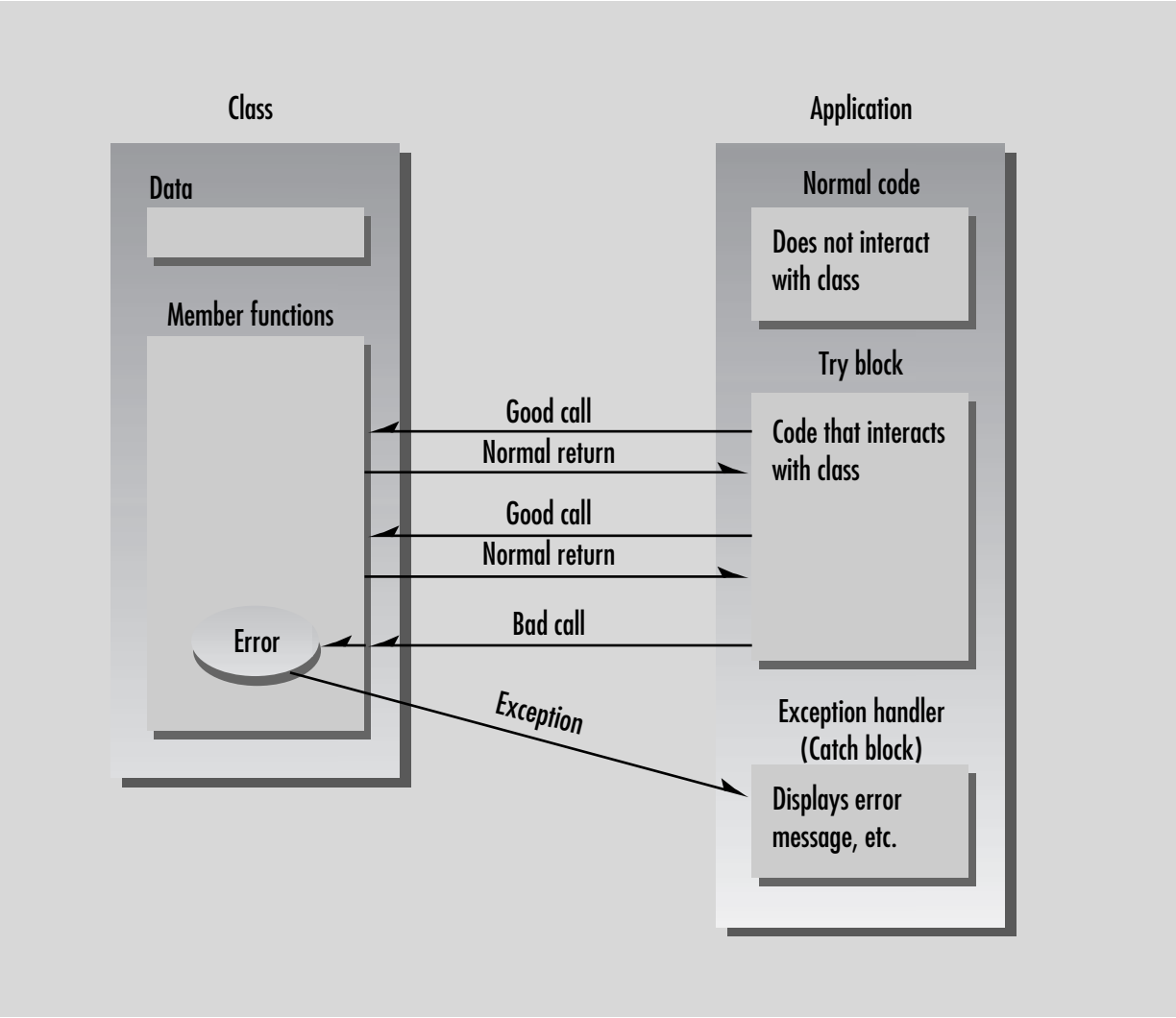
- Using exceptions makes code more readable and maintainable
  - Separate the core logic from error handling

```
if( somefunc() == ERROR_RETURN_VALUE )
   //handle the error or call error-handler function
else
   //proceed normally
if( anotherfunc() == NULL )
   //handle the error or call error-handler function
else
   //proceed normally
if( thirdfunc() == 0 )
   //handle the error or call error-handler function
else
   //proceed normally
```

Why Exceptions?

- In C++, there are many implicit calls and jumps which can cause errors.
  - Class constructor
  - Copy/move assignment
- A library function might encounter an error which is harder to communicate up the call-chain to the library user.
  - Communicating errors from deep within class libraries is the most important problem solved by exceptions.

Exception execution flow



```
const int MAX = 3;

class Stack
{
  private:
    int st[MAX];
    int top;

  public:
    class Range{ };
    Stack() {top = -1;}
    void push(int var)
    {
      if (top == MAX - 1)
        throw Range();
      st[++top] = var;
    }

    int pop()
    {
      if (top < 0)
        throw Range();
      return st[top--];
    }
};
```

Exception class, must be public

Generate exception

```
const int MAX = 3;

class Stack
{
  private:
    int st[MAX];
    int top;

  public:
    class Range{ };
    Stack() {top = -1;}
    void push(int var)
    {
      if (top == MAX - 1)
        throw Range();
      st[++top] = var;
    }

    int pop()
    {
      if (top < 0)
        throw Range();
      return st[top--];
    }
};
```

```
int main()
{
  Stack st;
  try
  {
    st.push(1);
    st.push(2);
    st.push(3);
    st.push(4);
    cout << st.pop() << endl;
    cout << st.pop() << endl;
    cout << st.pop() << endl;
    cout << st.pop() << endl;
  }
  catch(Stack::Range)
  {
    cout << "Exception:Stack full or empty\n";
  }
  cout << "here after catch\n";
}
```

Prints:

Exception: Stack full or empty
here after catch

```
const int MAX = 3;

class Stack
{
  private:
    int st[MAX];
    int top;

  public:
    class Range{ };
    Stack() {top = -1;}
    void push(int var)
    {
      if (top == MAX - 1)
        throw Range();
      st[++top] = var;
    }

    int pop()
    {
      if (top < 0)
        throw Range();
      return st[top--];
    }
};
```

```
int main()
{
  Stack st;
  try
  {
    st.push(1);
    st.push(2);
    st.push(3);
    //st.push(4);
    cout << st.pop() << endl;
    cout << st.pop() << endl;
    cout << st.pop() << endl;
    cout << st.pop() << endl;
  }
  catch(Stack::Range)
  {
    cout << "Exception:Stack full or empty\n";
  }
  cout << "here after catch\n";
}
```

Prints:

3 2 1
Exception:Stack full or empty
here after catch

```
const int MAX = 3;

class Stack
{
  private:
    int st[MAX];
    int top;

  public:
    class RangeFull{ };
    class RangeEmpty{ };
    Stack() {top = -1;}
    void push(int var)
    {
      if (top == MAX - 1)
        throw RangeFull();
      st[++top] = var;
    }

    int pop()
    {
      if (top < 0)
        throw RangeEmpty();
      return st[top--];
    }
};
```

```
int main()
{
  Stack st;
  try
  {
    st.push(1);
    st.push(2);
    st.push(3);
    st.push(4);
    cout << st.pop() << endl;
    cout << st.pop() << endl;
    cout << st.pop() << endl;
    cout << st.pop() << endl;
  }
  catch(Stack::RangeFull) {cout << "Exception:Stack full\n";}
  catch(Stack::RangeEmpty) {cout << "Exception:Stack empty\n";}

 cout << "here after catch\n";
}
```

Prints:

Exception:Stack full
here after catch

```
const int MAX = 3;

class Stack
{
  private:
    int st[MAX];
    int top;

  public:
    class RangeFull{ };
    class RangeEmpty{ };
    Stack() {top = -1;}
    void push(int var)
    {
      if (top == MAX - 1)
        throw RangeFull();
      st[++top] = var;
    }

    int pop()
    {
      if (top < 0)
        throw RangeEmpty();
      return st[top--];
    }
};
```

```
int main()
{
  Stack st;
  try
  {
    st.push(1);
    st.push(2);
    st.push(3);
    //st.push(4);
    cout << st.pop() << endl;
    cout << st.pop() << endl;
    cout << st.pop() << endl;
    cout << st.pop() << endl;
  }
  catch(Stack::RangeFull) {cout << "Exception:Stack full\n";}
  catch(Stack::RangeEmpty) {cout << "Exception:Stack empty\n";}

 cout << "here after catch\n";
}
```

Prints:

3 2 1
Exception:Stack empty
here after catch

```
const int MAX = 3;

class Stack
{
  private:
    int st[MAX];
    int top;

  public:
    class RangeFull{ };
    class RangeEmpty{ };
    Stack() {top = -1;}
    void push(int var)
    {
      if (top == MAX - 1)
        throw RangeFull();
      st[++top] = var;
    }

    int pop()
    {
      if (top < 0)
        throw RangeEmpty();
      return st[top--];
    }
};
```

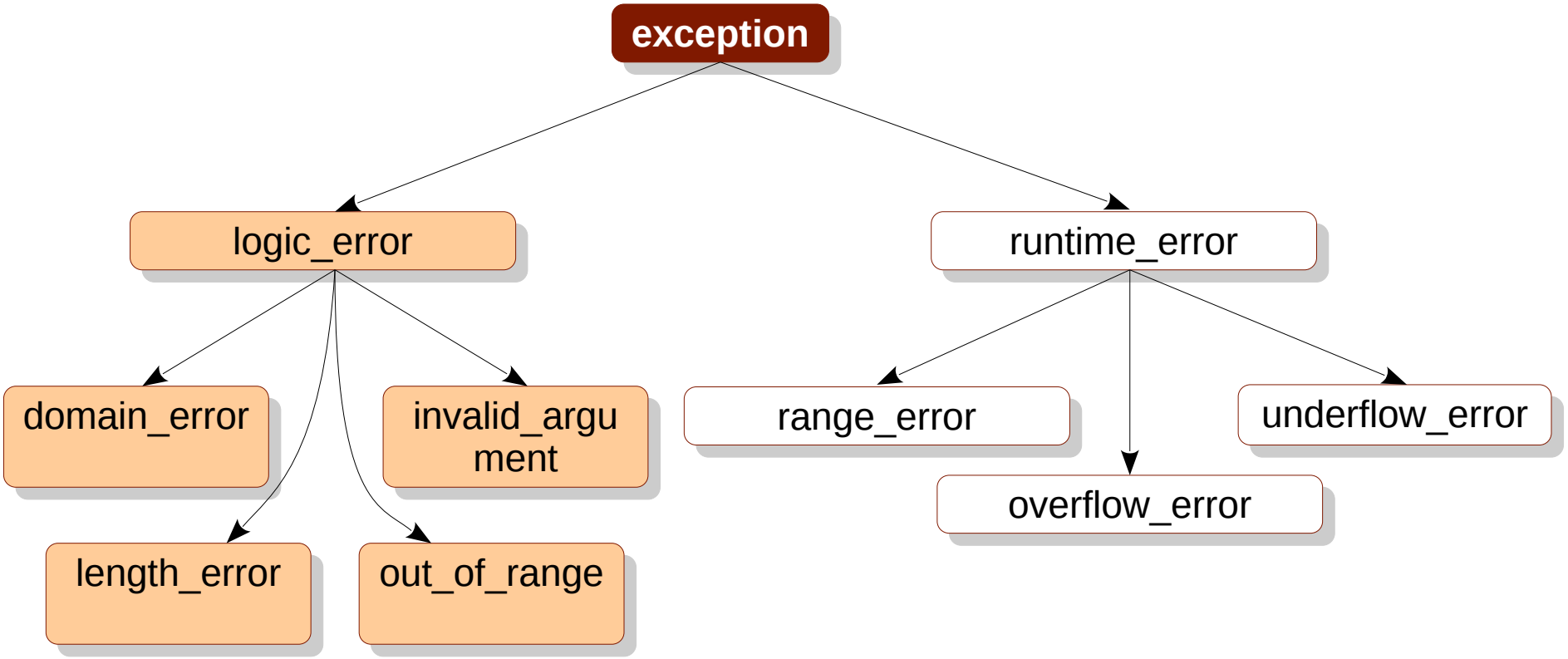
```
int main()
{
  Stack st;

  for (int i = 0; i < 6; i++)
  {
    try
    {
      st.push(i);
    }
    catch(Stack::RangeFull)
        {cout << "Exception:Stack full\n";}
    catch(Stack::RangeEmpty)
        {cout << "Exception:Stack empty\n";}
  }
  cout << "here after catch\n";
}
```

Prints:

Exception:Stack full
Exception:Stack full
Exception:Stack full
here after catch

C++ Exceptions



String exception

```
int main() {
  string s;

  try {
    //s.insert(0,"Hello");
    s.insert(1,"Hello");
  } catch (const std::exception& e) {
    cout << "The exception is caught." << endl;
  }

return 0;
}
```

Prints:
The exception is caught

Catching exceptions from calls

```
string s;
void fun() {
   s.insert(1,"Hello");
}

int main() {
  try {
    fun();
  }
  catch (const std::exception& e) {
    cout << "The exception is caught." << endl;
  }
  return 0;
}
```

Prints:
The exception is caught

Nested try-catch

```
string s;
void fun() {
  try {
      s.insert(1,"Hello");
  }
  catch(std::exception &e) {
    cout << "Caught in fun.\n";
  }
}
int main() {
  try {
    fun();
  } catch (const std::exception& e) {
    cout << "The exception is caught." << endl;
  }
  cout << "The program ends now.\n";

  return 0;
}
```

Prints:
Caught in fun.
The program ends now.

Catching and throwing

```
string s;
void fun() {
  try {
      s.insert(1,"Hello");
  } catch(std::exception &e) {
    cout << "Caught in fun.\n";
    throw e;
    //throw 5;
  }
}
int main() {
  try {
    fun();
  }
  catch (const std::exception& e) {
    cout << "The exception is caught in main.\n”;
  }
  catch (int x) {
    cout << "Int caught in main.\n";
  }
  cout << "The program ends now.\n";
}
```

Prints:

Caught in fun.
The exception is caught in main.
The program ends now.

Fin.