Αντικειμενοστραφής Προγραμματισμός

Μέρος 5ο: Χειρισμός εξαιρέσεων

Εξάμηνο Σπουδών: 6ο

Κωδικός Μαθήματος: 647

Τμήμα Μαθηματικών
Πανεπιστήμιο Ιωαννίνων

Μιχάλης Α. Μπέκος
bekos@uoi.gr

Περιεχόμενα

- Χειρισμός εξαιρέσεων
  - Κλάσεις εξαιρέσεις (exception classes)
  - Υλοποίηση κλάσεων εξαιρέσεων
  - Πολλαπλά `throws` και `catches`

- Προγραμματιστικές τεχνικές για τον χειρισμό εξαιρέσεων
  - Πότε επιβάλεται η δημιουργία εξαίρεσης
  - Ιεραρχίες κλάσεων εξαιρέσεων

Εισαγωγή

- Μια τυπική προσέγγιση στην ανάπτυξη προγραμμάτων είναι η εξής:
  - Αναπτύσσεται ένα πρόγραμμα υποθέτοντας ότι όλα θα πάνε όπως προβλέπεται
  - Έτσι, ολοκληρώνεται ο "πυρήνας" του προγράμματος
  - Στο τέλος, αντιμετωπίζονται οι "εξαιρετικές" περιπτώσεις

- Στη C++ ο χειρισμός των εξαιρέσεων γίνεται ως εξής:
  - Αρχικά, εντοπίζονται οι "εξαιρέσεις" (π.χ. διαίρεση με το μηδέν)
  - Ένας μηχανισμός "σηματοδοτεί" ότι κάτι ασυνήθιστο συμβαίνει
  - Ένα άλλο σημείο στον κώδικα "χειρίζεται" την εξαίρεση

Βασικά στοιχεία χειρισμού εξαιρέσεων

- Οι εξαιρέσεις υπονοείται ότι πρέπει να χρησιμοποιούνται με φειδώ
  - σε ειδικές περιπτώσεις

- Τα βασικά στοιχεία χειρισμού εξαιρέσεων εξηγούνται καλύτερα με παραδείγματα
  - ωστόσο, δύσκολα διδάσκονται μεγάλα παραδείγματα

- Η προσέγγιση μας θα είναι η εξής:
  - εξετάζουμε απλά παραδείγματα, στα οποία ουσιαστικά δεν χρειάζεται χειρισμός εξαιρέσεων
  - ωστόσο, στο μυαλό μας θα έχουμε πώς αυτά θα μπορούσαν να εφαρμοστούν σε πιο απαιτητικές περιπτώσεις

Ένα απλό παράδειγμα

- Θεωρείστε τον παρακάτω κώδικα

- Παρατηρήστε ότι ο κώδικας έχει πρόβλημα, αν η μεταβλητή `milk` πάρει την τιμή μηδέν (διαίρεση με το μηδέν)
  - ο κώδικας αγνοεί το πρόβλημα που δυνητικά μπορεί να υπάρξει
  - είναι και αυτός ένας τρόπος χειρισμού εξαιρέσεων (όχι όμως ο ενδεδειγμένος)

```
int main() {
    int donuts, milk;
    cout << "Enter number of donuts:";
    cin >> donuts;
    cout << "Enter number of glasses of milk:";
    cin >> milk;
    double dpg = donuts/static_cast<double>(milk);
    cout << "You have " << dpg << " donuts for each glass of milk." << endl;
}
```

Ένας πρώτος τρόπος χειρισμού της εξαιρετικής περίπτωσης

- Μια απλή λύση στο πρόβλημα του απλού παραδείγματος που εντοπίσαμε είναι μια εντολή ροής προγράμματος `if-else`

- Ο τρόπος αυτός δε συνιστά χειρισμό εξαίρεσης (με τον ορισμό που θα δούμε σε λίγο)
  - αποτελεί ωστόσο λύση στο πρόβλημα

```
int main() {
    int donuts, milk;
    cout << "Enter number of donuts:";
    cin >> donuts;
    cout << "Enter number of glasses of milk:";
    cin >> milk;
    if (milk <= 0)
      cout << donuts << " donuts and no milk." << endl;
    else {
        double dpg = donuts/static_cast<double>(milk);
        cout << "You have " << dpg << " donuts for each glass of milk." << endl;
    }
}
```

Hands-on ενότητα 1

Υλοποίηση μιας απλής προσέγγισης χειρισμού εξαίρεσης για το απλό παράδειγμα

Το πρώτο μας παράδειγμα χειρισμού εξαίρεσης

- Ο χειρισμός εξαιρέσεων στη C++ περιλαμβάνει τρία μπλοκ:
  - `try`
  - `throw`
  - `catch`

```
int main() {
  int donuts, milk;
  try {
    cout << "Enter number of donuts: ";          cin >> donuts;
    cout << "Enter number of glasses of milk: "; cin >> milk;

    if (milk <= 0) throw donuts;

    double dpg = donuts/static_cast<double>(milk);
    cout << "You have " << dpg << " donuts for each glass of milk." << endl;
  }
  catch(int e) {
    cout << e << " donuts, and no milk. Go buy some milk." << endl;
  }
  return 0;
}
```

Παρατηρήσεις επί της υλοποίησης

- Ο κώδικας μεταξύ των `try` και `catch` είναι ο ίδιος όπως και πριν
  - ωστόσο, το σώμα της `if` είναι απλούστερο: `if ( milk <= 0 ) throw donuts;`
  - καθαρότερος κώδικας
- Αν η μεταβλητή `milk` δεν είναι θετική, δηλώνεται ότι πρέπει να γίνει "κάτι κατ' εξαίρεση"
- Αυτό το "κάτι κατ' εξαίρεση" ορίζεται μετά τη λέξη κλειδί `catch`
- Το μπλοκ `try` χειρίζεται την "κανονική κατάσταση", ενώ το `catch` την "κατ' εξαίρεση κατάσταση"
- Διαχωρίζεται έτσι ο κανονικός από τον κατ' εξαίρεση κώδικα
  - Βέβαια για το απλό παράδειγμα που εξετάζουμε η επίπτωση δεν είναι σημαντική, αλλά στη γενική περίπτωση ο μηχανισμός είναι εξαιρετικά χρήσιμος

Τα μπλοκ `try` και `catch`

- Το μπλοκ `try` περιέχει τον κώδικα της κανονικής “κατάστασης”, όταν όλα πηγαίνουν καλά
- Όταν συμβεί κάτι ασυνήθιστο μέσα στο μπλοκ `try`, δημιουργείται εξαίρεση
  - Μετά τη λέξη κλειδί `throw` ακολουθεί ο τύπος της εξαίρεσης

```
int main() {
  int donuts, milk;
  try {
    cout << "Enter number of donuts: ";           cin >> donuts;
    cout << "Enter number of glasses of milk: "; cin >> milk;

    if (milk <= 0) throw donuts;

    double dpg = donuts/static_cast<double>(milk);
    cout << "You have " << dpg << " donuts for each glass of milk." << endl;
  }
  catch(int e) {
    cout << e << " donuts, and no milk. Go buy some milk." << endl;
  }
  return 0;
}
```

Τα μπλοκ `try` και `catch`

- Όταν δημιουργείται εξαίρεση, ο κώδικας μεταφέρεται από το μπλοκ `try` στο μπλοκ `catch`
  - Η εκτέλεση του μπλοκ `catch` λέγεται χειρισμός της εξαίρεσης
- Για κάθε εξαίρεση θα πρέπει να υπάρχει catch μπλοκ που την “χειρίζεται”

```
int main() {
  int donuts, milk;
  try {
    cout << "Enter number of donuts: ";          cin >> donuts;
    cout << "Enter number of glasses of milk: "; cin >> milk;

    if (milk <= 0) throw donuts;

    double dpg = donuts/static_cast<double>(milk);
    cout << "You have " << dpg << " donuts for each glass of milk." << endl;
  }
  catch(int e) {
    cout << e << " donuts, and no milk. Go buy some milk." << endl;
  }
  return 0;
}
```

Τα μπλοκ `try` και `catch`

- Στο παράδειγμα, το μπλοκ `catch` μοιάζει με τον ορισμό μιας συνάρτησης με παράμετρο `int`
  - δεν πρόκειται για συνάρτηση, ωστόσο λειτουργεί παρόμοια
- Η δημιουργία εξαίρεσης είναι σαν μια “κλήση συνάρτησης”

```
int main() {
  int donuts, milk;
  try {
    cout << "Enter number of donuts: ";           cin >> donuts;
    cout << "Enter number of glasses of milk: "; cin >> milk;

    if (milk <= 0) throw donuts;

    double dpg = donuts/static_cast<double>(milk);
    cout << "You have " << dpg << " donuts for each glass of milk." << endl;
  }
  catch(int e) {
    cout << e << " donuts, and no milk. Go buy some milk." << endl;
  }
  return 0;
}
```

Τα μπλοκ `try` και `catch`

- Στο παράδειγμα, η παράμετρος `e` ονομάζεται παράμετρος του μπλοκ `catch`
  - Κάθε μπλοκ `catch` έχει ακριβώς μια τέτοια παράμετρο
- Ο τύπος της παραμέτρου καθορίζει το είδος των εξαιρέσεων που μπορεί να χειριστεί το μπλοκ, ενώ το όνομα αυτής υποστηρίζει πρόσβαση στην εξαίρεση.

```
int main() {
  int donuts, milk;
  try {
    cout << "Enter number of donuts: ";           cin >> donuts;
    cout << "Enter number of glasses of milk: "; cin >> milk;

    if (milk <= 0) throw donuts;

    double dpg = donuts/static_cast<double>(milk);
    cout << "You have " << dpg << " donuts for each glass of milk." << endl;
  }
  catch(int e) {
    cout << e << " donuts, and no milk. Go buy some milk." << endl;
  }
  return 0;
}
```

Κλάσεις εξαιρέσεις

- Η δήλωση `throw` μπορεί να δημιουργήσει μια εξαίρεση οποιοδήποτε τύπου
- Μια κλάση εξαίρεσης περιέχει πληροφορίες σε σχέση με την εξαίρεση που δημιουργείται
  - για την ακρίβεια, το αντικείμενο που παράγεται κατά τη δημιουργία της εξαίρεσης τις περιέχει

```
int main() {
  int donuts, milk;
  try {
    cout << "Enter number of donuts: ";          cin >> donuts;
    cout << "Enter number of glasses of milk: "; cin >> milk;

    if (milk <= 0) throw NoMilk(donuts);

    double dpg = donuts/static_cast<double>(milk);
    cout << "You have " << dpg << "donuts for each glass of milk.";
  }
  catch(NoMilk e) {
    cout << e.getCount()  << " donuts, and no milk. Go buy some milk.";
  }
  return 0;
}
```

```
class NoMilk
{
public:
  NoMilk() {
    count = 1;
  }
  NoMilk(int howMany) {
    count = howMany;
  }
  int getCount() const {
    return count;
  }

private:
  int count;
};
```

Κλάσεις εξαιρέσεις

- Στο παράδειγμα, κατά τη δημιουργία της εξαίρεσης:
  - η δήλωση `throw` καλεί τον κατασκευαστή της κλάσης `NoMilk`
  - και παράγει ένα αντικείμενο της κλάσης `NoMilk`

```
int main() {
  int donuts, milk;
  try {
    cout << "Enter number of donuts: ";          cin >> donuts;
    cout << "Enter number of glasses of milk: "; cin >> milk;

    if (milk <= 0) throw NoMilk(donuts);

    double dpg = donuts/static_cast<double>(milk);
    cout << "You have " << dpg << "donuts for each glass of milk.";
  }
  catch(NoMilk e) {
    cout << e.getCount()  << " donuts, and no milk. Go buy some milk.";
  }
  return 0;
}
```

```
class NoMilk
{
public:
  NoMilk() {
    count = 1;
  }
  NoMilk(int howMany) {
    count = howMany;
  }
  int getCount() const {
    return count;
  }

private:
  int count;
};
```

Πολλαπλά `throws` και `catches`

- Ένα μπλοκ try δυνητικά μπορεί να δημιουργήσει περισσότερες από μια εξαιρέσεις
  - καθεμία από τις οποίες μπορεί να είναι διαφορετικού τύπου
  - π.χ., `DivideByZero`, `IndexOutOfBounds`, κ.ο.κ.

- Ωστόσο, το πολύ μια εξαίρεση δημιουργείται στο `try` μπλοκ
  - καθώς η δήλωση `throw` τερματίζει το μπλοκ `try`
  - και μεταφέρει τον έλεγχο στο αντίστοιχο `catch` μπλοκ

- Αντίστοιχα, κάθε μπλοκ `catch` χειρίζεται μόνο “έναν τύπο εξαίρεσης”
  - είναι σύνηθες να τοποθετούμε πολλαπλά μπλοκ `catch` μετά από κάθε μπλοκ `try`
  - για το χειρισμό “όλων των πιθανών εξαιρέσεων” που μπορεί να δημιουργηθούν

Hands-on ενότητα 2

Υλοποίηση ενός απλού παραδείγματος με πολλαπλά `throws` και `catches`

Τετριμμένες κλάσεις εξαιρέσεις

- Μια τετριμμένη κλάση εξαίρεσης δε συσχετίζεται με κάποια τιμή. Οπότε:
  - δεν έχει μεταβλητές μέλη
  - δεν έχει συναρτήσεις μέλη (εκτός του προκαθορισμένου κατασκευαστή)
- Ουσιαστικά ορίζει μόνο το όνομα της
  - χρησιμοποιείται απλά για τον χειρισμό της εξαίρεσης στο `catch` μπλοκ
- Παράδειγμα:

```
/**
 * Exception thrown when there is an attempt to
 * divide an integral or decimal value by zero.
 */
class DivideByZero {
//No member variables or functions needed.
};
```

Εξαιρέσεις και κληρονομικότητα

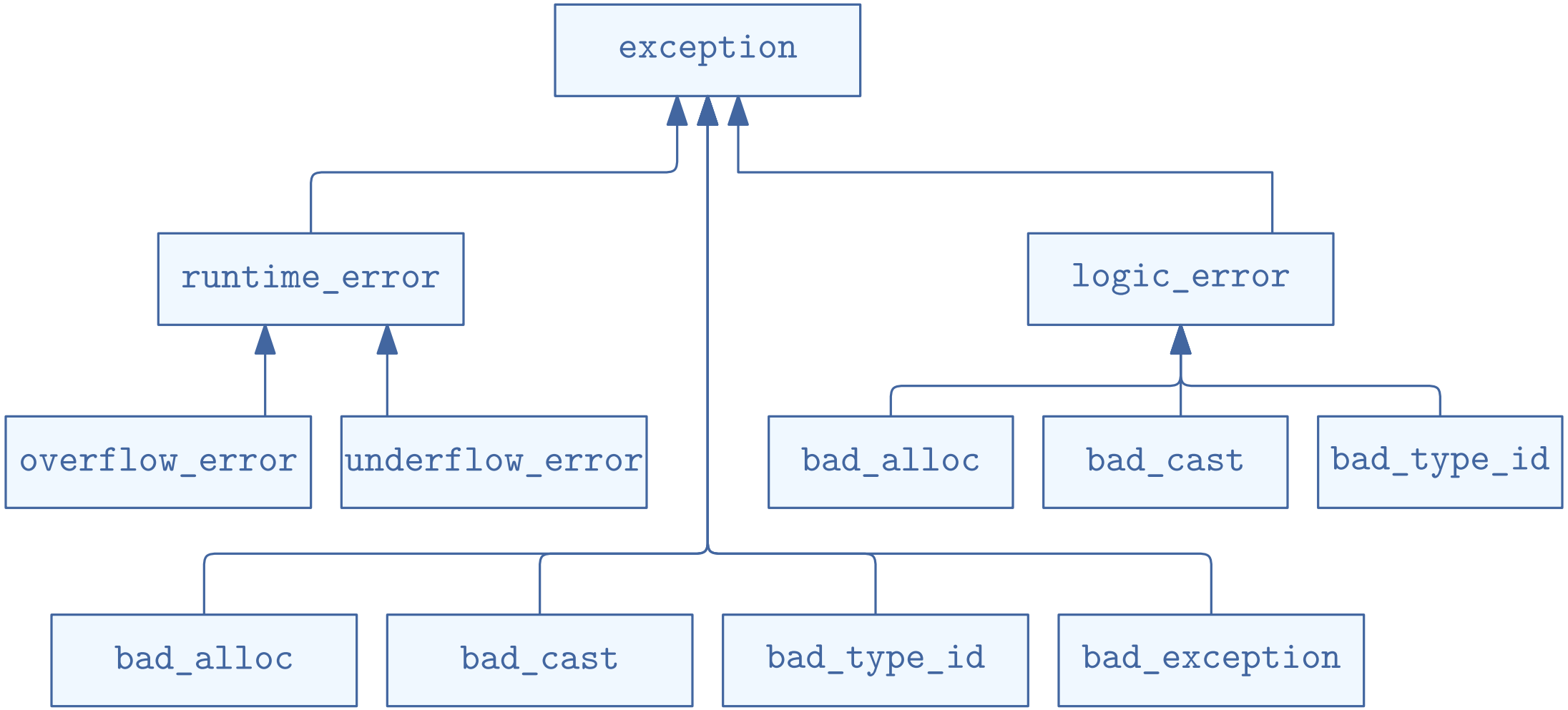
- Οι κλάσεις εξαιρέσεις συνήθως υλοποιούνται ως παράγωγες κλάσεις υπαρχόντων κλάσεων.
- Στη C++, η βασική κλάση είναι η `exception`
  - περιέχει την εικονική συνάρτηση `what` για την αποθήκευση μηνυμάτων σφάλματος
- Παράγωγες κλάσεις της `exception`
  - `runtime_error` – δημιουργείται από σφάλματα κατά τη διάρκεια της εκτέλεσης
  - `bad_alloc` – δημιουργείται κατά τη δημιουργία αντικειμένου από το `new`
  - `bad_cast` – δημιουργείται κατά τη μετατροπή δείκτη από την `dynamic_cast`

```
int main() {
  float num = 12.5, den = 0.0;
  try {
    if (den == 0) throw DivideByZero();
    cout << "Quotient is: " << num/den;
  }
  catch (DivideByZero& e)
    cout << "Exception: " << e.what();
}
```

```
#include <stdexcept>
using namespace std;

class DivideByZero : public runtime_error {
public:
  // Message passed to the runtime_error class
  Exception() : runtime_error("Division by zero") {}
};
```

Η ιεραρχία εξαιρέσεων της Standard Library



Hands-on ενότητα 3

Υλοποίηση παραδείγματος χειρισμού πιθανής διαίρεσης με το μηδέν

Συναρτήσεις που δημιουργούν εξαιρέσεις

- Μια συνάρτηση μπορεί να δημιουργήσει μια εξαίρεση
- Διαφορετικά σημεία του προγράμματος από τα οποία καλείται η συνάρτηση μπορούν να χειριστούν την εξαίρεση με διαφορετικούς τρόπους
- Ο κώδικας που καλεί τη συνάρτηση τοποθετείται σε `try` μπλοκ
- Αντίστοιχα, ο χειρισμός της εξαίρεσης γίνεται σε `catch` μπλοκ μετά το `try` μπλοκ

```
int main() {
  int x = 12.5, y = 0.0;

  try {
   cout << "Result: " << divide(x, y);
  }
  catch(DivideByZero) {
   cout << "Error: Division by zero!";
   exit(0);
  }
}
```

```
class DivideByZero {};

double divide(int x, int y) throw (DivideByZero);

double divide(int x, int y) throw (DivideByZero) {
  if (y == 0) {
    throw DivideByZero();
  }
  return x/static_cast<double>(y);
}
```

Συναρτήσεις που δημιουργούν εξαιρέσεις

- Οι συναρτήσεις που δημιουργούν εξαιρέσεις θα πρέπει να ”προειδοποιήσουν” το χρήστη ότι ίσως δημιουργήσουν τέτοιες εξαιρέσεις
- Αυτό επιτυγχάνεται με την απαρίθμηση τους τόσο στη δήλωση όσο και στην υλοποίηση
- Το τμήμα στο οποίο γίνεται η απαρίθμηση λέγεται τμήμα καθορισμού εξαιρέσεων ή throw list
- Παράδειγμα: `void someFunction() throw(DivideByZero, OtherException);`

```
int main() {
  int x = 12.5, y = 0.0;

  try {
   cout << "Result: " << divide(x, y);
  }
  catch(DivideByZero) {
   cout << "Error: Division by zero!";
   exit(0);
  }
}
```

```
class DivideByZero {};

double divide(int x, int y) throw (DivideByZero);

double divide(int x, int y) throw (DivideByZero) {
  if (y == 0) {
    throw DivideByZero();
  }
  return x/static_cast<double>(y);
}
```

Καθορισμός των εξαιρέσεων

- Εάν μια συνάρτηση δημιουργήσει μια εξαίρεση, η οποία δεν περιλαμβάνεται στη λίστα καθορισμού των εξαιρέσεων της:
  - δεν υπάρχει σφάλμα (μεταγλωττιστή ή κατά την εκτέλεση)
- Καλείται αυτόματα η συνάρτηση `upexpected()`
  - η προκαθορισμένη υλοποίηση της είναι ο τερματισμός του προγράμματος
  - δεν απαιτεί ειδικά includes ή κάποιο ειδικό όνομα χώρου
  - ωστόσο, η υλοποίηση αυτή μπορεί να τροποποιηθεί (πέρα από το σκοπό του μαθήματος)
- Ίδιο είναι το αποτέλεσμα εάν δεν βρεθεί `catch` μπλοκ για το χειρισμό μιας εξαίρεσης (η οποία μπορεί και να περιλαμβάνεται στη λίστα καθορισμού εξαιρέσεων)

```
//Exception types DivideByZero and OtherException
//treated normally. All others invoke unexpected()
void someFunction() throw(DivideByZero, OtherException);
```

```
//Empty exception list
//All exceptions invoke unexpected()
void someFunction() throw();
```

Παράγωγες κλάσεις εξαιρέσεις

- Υπενθύμιση: Τα αντικείμενα μιας παράγωγης κλάσης είναι επίσης αντικείμενα της βασικής
- Ας υποθέσουμε το εξής σενάριο:
  - η κλάση `B` είναι παράγωγη της `A`
- Αν η κλάση `A` είναι στο τμήμα καθορισμού εξαιρέσεων μιας συνάρτησης,
  - τότε και οι εξαιρέσεις τύπου `B` θα διαχειριστούν (ως τύπου `A`)
- Σημείωση: Δεν υποστηρίζονται αυτόματες μετατροπές τύπων
  - π.χ., οι εξαιρέσεις τύπου `double` είναι διαφορετικές από αυτές τύπου `int`

Καλές και κακές πρακτικές

- Καλή πρακτική: Κάθε εξαίρεση πρέπει να διαχειρίζεται
  - διαφορετικά, ενδέχεται να τερματίσει το πρόγραμμα
  - πράγματι, αν η εξαίρεση δημιουργηθεί, τότε θα κληθεί η συνάρτηση `unexpected()`
  - η οποία με τη σειρά της θα καλέσει τη συνάρτηση `terminate()`

- Κακή πρακτική: Κατάχρηση των εξαιρέσεων
  - οι εξαιρέσεις αλλάζουν τη ροή του προγράμματος
  - συνεπώς, θα πρέπει να χρησιμοποιούνται με μέτρο

- Καλή πρακτική: Εάν θεωρείτε ότι κάποιο σημείο στον κώδικα απαιτεί δημιουργία εξαίρεσης
  - εξετάστε το ενδεχόμενο να γράψετε τον κώδικα σας χωρίς αυτή
  - εάν κάτι τέτοιο δεν είναι εφικτό, τότε προχωρήστε στη υλοποίησή της

Επαναδημιουργία μιας εξαίρεσης

- Μια εξαίρεση μπορεί να δημιουργηθεί και σε ένα `catch` μπλοκ
  - σε σπάνιες περιπτώσεις
- Για την ακρίβεια, μπορεί να δημιουργηθεί:
  - μια νέα εξαίρεση ή
  - ακόμη και η ίδια εξαίρεση
    (με αυτή που χειρίζεται το `catch` μπλοκ)

```
#include <iostream>
using namespace std;

int main() {
  try {
    int a, b;
    cout << "Enter two integer values: ";
    cin >> a >> b;
    try {
      if(b == 0) {
        throw b;
      }
      else {
        cout << a/static_cast<double>(b);
      }
    }
    catch(int e) {
      throw e; //rethrowing the exception
    }
  }
  catch(int) {
    cout << "Division by zero";
  }
  return 0;
}
```

Hands-on ενότητα 4

Υλοποίηση παραδείγματος ανάγνωσης δεδομένων από αρχείο

Σύνοψη

- Ο χειρισμός των εξαιρέσεων επιτρέπει τον διαχωρισμό μεταξύ των “κανονικών” και των “εξαιρετικών” περιπτώσεων στον κώδικα
- Οι εξαιρέσεις δημιουργούνται σε `try` μπλοκ
  - ή μέσω συναρτήσεων των οποίων η κλήση γίνεται σε ένα `try` μπλοκ
- Ο χειρισμός των εξαιρέσεων γίνεται σε `catch` μπλοκ
- Κάθε `try` μπλοκ συνήθως ακολουθείται από ένα ή περισσότερα `catch` μπλοκ
  - οι πιο συγκεκριμένες εξαιρέσεις διαχειρίζονται πρώτες
- Εξαιρέσεις που δεν διαχειρίζονται οδηγούν σε τερματισμό του προγράμματος
- Καλή πρακτική: Χρήση των εξαιρέσεων με μέτρο