CS 2230
CS II: Data structures
Meeting 20: exceptions
Brandon Myers
University of Iowa
Today’s big ideas

• In Java, errors while the program is running are typically indicated with Exceptions

• Exceptions are either
  • “checked” (you have to declare them)
  • or, “unchecked” (you don’t have to declare them)

• The Java API is helpful
You’ve seen a lot of *exceptions* already

e.g., from HW 3:
Peer instruction

What does this message mean?

a) JUnit has an internal bug
b) there are 42 errors in LinkedList.java
c) line 54 of LinkedListTest.java tried to use dot (.) on an object reference that was null
d) testRemoveGetAppend was expecting the answer 54 but got 42
e) line 42 of LinkedList.java tried to use dot (.) on an object reference that was null

https://b.socrative.com/login/student/
CS2230A ids 1000-4999
CS2230B ids 5000+
You’ve seen a lot of exceptions already

e.g., from HW 3:

- a message from JUnit
- name of the exception
- “stack trace”, or “call stack”
  the sequence of unreturned method calls from most recent to oldest
Handling errors using exceptions

```java
void say(String words) {
    if (message == null) {
        throw new IllegalArgumentException("I don't like nulls");
    } else {
        System.out.println(words);
    }
}
```

- An exception is just an object! Create one like any other object.
- `throw` tells Java you want to “sound the alarm”. There will be special handling that happens next...
- **catch the exception**: if there is code to react to the exception, it will run
- if there is no such code, then the program will crash
Aside: a different approach

read_file might encounter bad situations

Code that uses read_file

```java
public static void main(String[] args) {
    byte[] contents = new byte[5000000];
    String filename = args[1];
    int error = read_file(filename, b);
    if (error != 0) {
        if (error == FILE_NOT_FOUND) {
            System.out.println("Can't find it");
        } else if (error == FILE_IS_A_DIRECTORY) {
            System.out.println("Not a file");
        } else if (error == FILE_FORMAT_UNRECOGNIZED) {
            System.out.println("Can't read it");
        }
    } else {
        // everything is fine
        // do stuff with contents
    }
}
```

```java
int read_file(String filename, byte[] result) {
    File f = open(filename);
    if (f == null) {
        return FILE_NOT_FOUND;
    }

    if (!isFile(f)) {
        return FILE_IS_DIRECTORY;
    }

    int position = 0;
    position += num_bytes;
    while (num_bytes != 0) {
        int num_bytes = read(f, position, result);
        if (num_bytes < 0) {
            return FILE_FORMAT_UNRECOGNIZED;
        }
        position += num_bytes;
    }

    return SUCCESS;
}
```

// error codes
int SUCCESS = 0;
int FILE_NOT_FOUND = 1;
int FILE_IS_DIRECTORY = 2;
int FILE_FORMAT_UNRECOGNIZED = 3;
The Java approach

void read_file(String filename, byte[] result) /*more stuff here...*/ {
    File f = open(filename);
    int position = 0;
    position += num_bytes;
    while (num_bytes != 0) {
        position += read(f, position, result);
    }
}  

try-catch block

If code in try {} throws an exception then one of the

catch (...) {} blocks will handle it depending on the type of exception

After catch is done, the program will continue (unless you exit the program, as shown in the example)
public static void main(String[] args) {
    int x = 0;
    int y = 1;
    try {
        if (x == y) {
            throw new ABCException("Attack of the ABC");
        } else {
            throw new XYZException("Attack of the XYZ");
        }
    } catch (ABCException e) {
        System.out.println("I caught ABC");
    } catch (XYZException e) {
        System.out.println("I caught XYZ");
    }
    System.out.println("All done");
}

What does the program print?

a) All done
b) Attack of the ABC
c) Attack of the XYZ
d) I caught ABC
   All done
e) I caught XYZ
   All done
Declare your (checked) exceptions

```java
import java.io.IOException;

class PlayWithExceptions {
    public static void doom() {
        throw new IOException("haha");
    }

    public static void main(String[] args) {
        doom();
    }
}

compile error
```

```text
exception_checked.java:6: error: unreported exception IOException;
    must be caught or declared to be thrown
    throw new IOException("haha");
   ^
1 error
```
Declare your (checked) exceptions

```java
import java.io.IOException;

class PlayWithExceptions {
    public static void doom() throws IOException {
        throw new IOException("haha");
    }

    public static void main(String[] args) {
        doom();
    }
}

compile error
```

```text
exception_checked.java:10: error: unreported exception IOException; must be caught or declared to be thrown
    doom();
    ^
1 error
```
Declare your (checked) exceptions

```java
import java.io.IOException;

class PlayWithExceptions {
    public static void doom() throws IOException {
        throw new IOException("haha");
    }

    public static void main(String[] args) {
        try {
            doom();
        } catch (IOException e) {
            System.out.println("phew, caught it");
        }
    }
}

compiler is happy now
Wait...why didn’t we have to write that in HW3?

```java
public Object get(int i) throws NullPointerException {
    ListNode current = getIthListNode(i);

    // Could throw NullPointerException
    // because we are using dot
    return current.getData();
}
```
Inheritance tree for Java exceptions

Exceptions are “checked” except for

RuntimeExceptions are “unchecked”
public static void foo(int x, y) {
    try {
        if (x == y) {
            throw new IOException("Attack of the ABC");
        } else {
            Object o = null;
            o.toString();
        }
    } catch (IOException e) {
        System.out.println("I caught ABC");
    }
}

(Assuming foo is within a class) What is the result of compiling this code?

a) compile error: need to declare or catch IOException
b) compile error: need to declare or catch NullPointerException
c) compiles successfully
d) compile error: cannot catch IOException
e) compile error if x!=y
Java API

https://docs.oracle.com/javase/8/docs/api/java/lang/Throwable.html
Today’s big ideas

• In Java, errors while the program is running are typically indicated with Exceptions

• Exceptions are either
  • “checked” (you have to declare them)
  • or, “unchecked” (you don’t have to declare them)

• The Java API is helpful