CS 2230

CS II: Data structures

Meeting 20: generic types, exceptions, higher order functions

Brandon Myers

University of Iowa
Today’s big ideas

• You can use **generic types** to write a data structure once and have it work for any element type

• 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)

• *Higher order functions* are another way to reuse code and provide a separation of concerns
Generic types: time to stop casting

Wizard by Wes Breazell from the Noun Project
Scrambled: Match the code to the comment

// 1
DogList x = DogLinkedList();
x.add(new Dog("pug"));
Dog element = x.get(0);
CatList y = CatLinkedList();
y.add(new Cat("Tabby"));
Cat element2 = y.get(0);

// 2
List x = LinkedList();
x.add(new Dog("pug"));
Dog element = x.get(0);
List y = LinkedList();
y.add(new Cat("Tabby"));
Cat element2 = y.get(0);

// 3
List x = LinkedList()
x.add(new Dog("pug"));
Dog element = (Dog) x.get(0);
List y = LinkedList()
y.add(new Cat("tabby"));
Cat element2 = (Cat) y.get(0);

// 4
List<Dog> x = LinkedList<>()
x.add(new Dog("pug"));
Dog element = x.get(0);
List<Cat> y = LinkedList<>()
y.add(new Cat("tabby"));
Cat element = y.get(0);

// A. does not compile

// B. compiles but poor code

// C. compiles and good code

https://b.socrative.com/login/student/
room CS2230X ids 1000-2999
room CS2230Y ids 3000+
Defining a classes/interfaces with a generic type

```java
interface List<T> {
    T get(int i);
    T remove(int i);
    void append(T element);
}
```

```java
class LinkedList<T> implements List<T> {
    private ListNode<T> header;

    private class ListNode<T> {
        public T data;
        public ListNode<T> next;
    }
    T get(int i) { ... }
    T remove(int i) { ... }
    void append(T element) { ... }
}
```

List<T> really represents an “infinite” number of interfaces:
List<Integer>
List<Cat>
List<Dog>
List<String>...
Compile errors / runtime errors

A) code has compile error
B) code has runtime error
C) code is fine

// 1
List<Dog> l = new ArrayList<Dog>();
l.add(new Dog());
Object d = l.get(0);
// 2
List<Object> l = new ArrayList<Object>();
l.add(new Dog());
Dog d = l.get(0);
// 3
List<Object> l = new ArrayList<Object>();
l.add(new Dog());
Dog d = (Dog) l.get(0);
// 4
List<Dog> l = new ArrayList<Dog>();
l.add(new Object());
Object d = l.get(0);
// 5
List<Object> l = new ArrayList<Object>();
l.add(new Object());
Dog d = (Dog) l.get(0);

https://b.socrative.com/login/student/
room CS2230X ids 1000-2999
room CS2230Y ids 3000+
Exception

an event indicating that something exceptional (usually bad) has happened

all exceptions are either

handled

or crash the program
You’ve seen a lot of exceptions already
e.g., from HW 3:

```
java.lang.NullPointerException
at hw3.LinkedList.append(LinkedList.java:42)
at hw3.LinkedListTest.testRemoveGetAppend(LinkedListTest.java:54)
```
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/
room CS2230X ids 1000-2999
room CS2230Y ids 3000+
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);
    }
}
```

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: an alternative to exceptions

`read_file` might encounter bad situations

```java
// read bytes from the file into result array.
// Return an error code or success
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;

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
    }
}
```
The Java approach

read_file might encounter bad situations

```java
// read bytes from the file into result array.
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)

```java
public static void main(String[] args) {
    byte[] contents = new byte[5000000];
    String filename = args[1];
    try {
        read_file(filename, b);
    } catch (FileNotFoundException ex) {
        System.out.println("Can't find it");
        System.exit(1); // exit the program
    } catch (IOException ex) {
        System.out.println("Can't read it");
        System.exit(1); // exit the program
    }
}
```
```java
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

https://b.socrative.com/login/student/
room CS2230X ids 1000-2999
room CS2230Y ids 3000+
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
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

```
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
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

- Throwable
  - Error
  - Exception
    - RuntimeException
      - NullPointerException
      - ArrayOutOf BoundsException
    - IOException
  - X is a Y

X is Y
Inheritance tree for Java exceptions

- Exceptions are “checked”, i.e., compiler requires that you catch or throw it
- RuntimeExceptions are “unchecked”, i.e., do nothing, usually you just let it crash your program
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
Higher order functions

bigger lesson: code is data, too
Higher order functions and data structures

Often it is helpful for data structures to include higher order functions

for example:
• apply a function to every element of a list
• use a function to eliminate some elements
HW5 makes use of higher order functions, starting at the IntApply class in Part 2

why?

to separate

**Iteration**

dealing with elements

from **business logic**

doing the actual computation on an element

\[ \text{next, next, next, next, ...} \]

\[ \text{output} = 2 \times \text{input} \]
Higher order functions over lists

// call the function f on all elements in the list
void applyAll(IntegerFunction f);

(see ArrayList.java)
public interface TwoInputFunction {
    public int op(int x, int y);
}

class Sub implements TwoInputFunction {
    public int op(int x, int y) { return x - y; }
}

class Mult implements TwoInputFunction {
    public int op(int x, int y) { return x * y; }
}

class PeerInstruction {
    public static int compose(TwoInputFunction h, TwoInputFunction g, int x, int y) {
        return g.op(h.op(x, y), h.op(y, x));
    }
    public static void main(String[] args) {
        System.out.println(compose(new Sub(), new Sub(), 10, 14));
        System.out.println(compose(new Sub(), new Mult(), 10, 14));
    }
}

What is printed when we run PeerInstruction’s main() method?
Today’s big ideas

• You can use *generic types* to write a data structure once and have it work for any element type

• 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)

• *Higher order functions* are another way to reuse code and provide a separation of concerns