Correctness and Robustness

With correctness, problems are the fault of the program's (or module's) code.

- only need to check for the correct situation
- program code only needs to detect a problem
 - handling the problem (i.e. fixing the bug) is the programmer's job

With robustness, problems are the fault of something outside the program (or module).

- check for incorrect situations
- program code also needs to handle the problem
 - attempt to recover if possible e.g. prompt for new input
 - if recovery is not possible, avoid doing damage and display an informative message so the user can understand and attempt to fix the problem
 - crashing is not reasonable

Signaling Errors

CPSC 225: Intermediate Programming . Spring 2025

Two options -

int index = str.indexOf(','); if (index == -1) { ... } // handle error String before = str.substring(0,index);

108

- return a special value
 - error handling is part of the normal flow of control
 - not always possible
 - caller can ignore
- throw an exception
 - typically only used when the error truly is an error, not a normally expected outcome
 - can separate error handling from the normal flow of control
 - allows for streamlined error handling
 - simplify code by grouping error handing for a related block of code in one place
 - simplify code when execution of a block of code should not continue if there's a problem with one step
 - some exception types must be handled

```
CPSC 225: Intermediate Programming • Spring 2025
```

Implementing Robustness

Considerations -

• can the error be handled in the same place it is detected?

user doesn't enter a positive number error occurred so the caller can do something
--

CPSC 225: Intermediate Programming • Spring 2025

CPSC 225: Intermediate Programming • Spring 2025

Throwing Exceptions					
<pre>if (condition) { throw exception-object; }</pre>					
The thing being thrown is an object. – typically you create a new object rather than throwing one that already exists					
 contains info about what went wrong and where object type provides information about the problem which can be used in the program to determine how to handle the problem detail message provides more information for the programmer/user stack trace identifies where the exception was thrown 					
e.getMessage(); e.printStackTrace();					

Example

Types of Exceptions

CPSC 225: Intermediate Programming . Spring 202

CPSC 225: Intermediate Programming . Spring 2025

A class hierarchy groups types of exceptions by function.

- Throwable is the top-level class - two main subclasses: Error and Exception
- Error is used for serious, fatal errors something that there is no reasonable way to handle
 - e.g. problems with the Java VM
 - should not be caught nothing the program can do
- Exception is used for errors that can be handled
 - generally the only things you'll throw (and catch) are subclasses of Exception

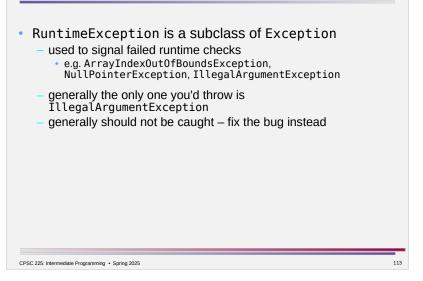
112

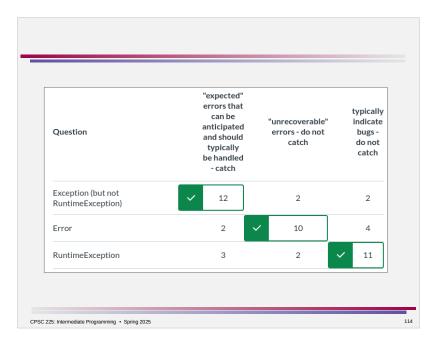
must be caught (unless subclass of RuntimeException)

```
public static Card read () throws ParseException {
 char ch = TextIO.getAnyChar();
if ( ch != ' ' ) {
    throw new ParseException("expected space; got "+ch);
  String word = TextIO.getWord();
  if ( !word.equals("of") ) {
    throw new ParseException("expected 'of'; got "+
                             value):
  char ch2 = TextI0.getAnyChar();
  if ( ch2 != ' ' ) {
    throw new ParseException("expected space; got "+ch2);
 String suit = TextIO.getWord();
if ( !suit.equals("hearts") && !suit.equals("diamonds") &&
       !suit.equals("spades") && !suit.equals("clubs") ) {
    throw new ParseException("invalid suit "+suit);
  return new Card(suit,value);
}
```

Types of Exceptions

CPSC 225: Intermediate Programming . Spring 202



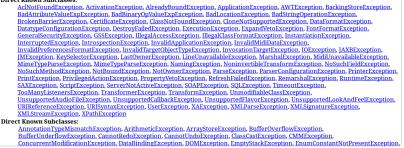


Catching Exceptions

A section of code that may throw an exception goes inside a *try block*.

One or more *catch blocks* (one for each type of exception) contain code to be executed if that type of exception is thrown.

Types of Exceptions Choose an appropriate exception type for your error. Java provides a number of exception types (though most are for very specific purposes) can also define your own subclass Exception, RuntimeException, or an existing exception type Direct Known Subclasses:



Catching Exceptions

When an exception is thrown, control immediately transfers to the nearest enclosing catch block.

- matching means the thrown object's type is the same or a subclass of the catch block's declared exception type
- if there is no matching catch within the current method, a matching catch is sought in the caller (and so forth)

Then -

CPSC 225: Intermediate Programming • Spring 2025

116

- the body of the catch block is executed
- the finally block is executed (if any)
- control continues immediately following that try-catch

117

CPSC 225: Intermediate Programming • Spring 2025

```
Int sum = 0;
for ( ; true ; ) {
   String input;
   System.out.print("please type something: ");
   input = scanner.nextLine();
   int number = Integer.parseInt(input);
   if ( number == 0 ) { break; }
   sum = sum+number;
}
System.out.println("the sum is: "+sum);
```

CPSC 225: Intermediate Programming • Spring 2025

```
10, 20, abc, 30, 0, 40, and xyz
Example
int sum = 0;
trv {
 for (; true ; ) {
    String input;
    System.out.print("please type something: ");
    input = scanner.nextLine();
    int number = Integer.parseInt(input);
    if ( number == 0 ) { break; }
    sum = sum+number;
  }
  System.out.println("the sum is: "+sum);
} catch ( NumberFormatException e ) {
  System.out.println("that wasn't a number!");
CPSC 225: Intermediate Programming · Spring 2025
```

```
Int sum = 0;
for ( ; true ; ) {
    String input;
    System.out.print("please type something: ");
    input = scanner.nextLine();
    try {
      int number = Integer.parseInt(input);
      if ( number == 0 ) { break; }
      sum = sum+number;
    } catch ( NumberFormatException e ) {
      System.out.println("that wasn't a number!");
    }
}
System.out.println("the sum is: "+sum);
```

CPSC 225: Intermediate Programming • Spring 2025

CPSC 225: Intermediate Programming • Spring 2025

```
10, 20, abc, 30, 0, 40, and xyz
Example
int sum = 0;
try {
  for ( ; true ; ) {
    String input;
    System.out.print("please type something: ");
    input = scanner.nextLine();
    int number = Integer.parseInt(input);
    if ( number == 0 ) { break; }
    sum = sum+number;
  }
} catch ( NumberFormatException e ) {
  System.out.println("that wasn't a number!");
} finallv {
  System.out.println("the sum is: "+sum);
}
```

121

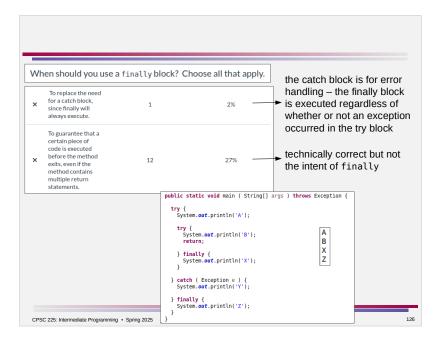
```
Example
int sum = 0:
try {
  for (; true ; ) {
    String input;
    System.out.print("please type something: ");
    input = scanner.nextLine();
    int number = Integer.parseInt(input);
if ( number == 0 ) { break; }
    sum = sum+number;
} catch ( NumberFormatException e ) {
  System.out.println("that wasn't a number!");
}
System.out.println("the sum is: "+sum);
CPSC 225: Intermediate Programming • Spring 2025
```

10, 20, abc, 30, 0, 40, and xyz

```
int[] numbers = new int[10];
for ( int index = 0 ; index < numbers.length ;</pre>
      index++ ) { numbers[index] = index; }
try {
 for ( ; true ; ) {
    String input:
    System.out.print("please type something: ");
    input = scanner.nextLine();
    try {
      int number = Integer.parseInt(input);
      if ( number == 0 ) { break; }
      System.out.println(numbers[number]);
    } catch ( NumberFormatException e ) {
      System.out.println("not a number!");
  }
} catch ( ArrayIndexOutOfBoundsException e ) {
    System.out.println("not a legal index!");
}
```

```
int[] numbers = new int[10];
-for ( int index = 0 ; index < numbers.length ;
       index++ ) { numbers[index] = index; }
 for ( ; true ; ) {
   String input;
   System.out.print("please type something: ");
   input = scanner.nextLine();
   try {
     int number = Integer.parseInt(input);
     if ( number == 0 ) { break; }
     System.out.println(numbers[number]);
   } catch ( NumberFormatException e ) {
     System.out.println("that wasn't a number!");
   } catch ( ArrayIndexOutOfBoundsException e ) {
     System.out.println("not a legal index!");
   }
 }
 CPSC 225: Intermediate Programming . Spring 202
```

Answer		Respondents	Percentage	
files, ne connec databa connec closed	twork tons, or	14	32%	
and pre program	iting due to Indled	2	5% —	 the catch block is for error handling – the finally block is executed regardless of whether or not an exception occurred in the try block
code th regardl whethe excepti such as		15	34%	



Not Catching Exceptions

You may choose not to catch/handle an exception in a method.

 e.g. the method can't solve the problem, but its caller might be able to

A method that throws an exception (other than a RuntimeException) without catching it must indicate that.

 — @exception tag in javadoc comments identifying exception type and when it occurs

128

throws declaration in method header

Caller's Responsibility

When the problem cannot be handled in the place it was detected, the caller has two options for handling -

- look before you leap
 - check for problematic conditions and only call the method if it can succeed
- leap before you look
 - call the method and deal with the error when it is signaled

Generally look first if you can. Prefer leaping if -

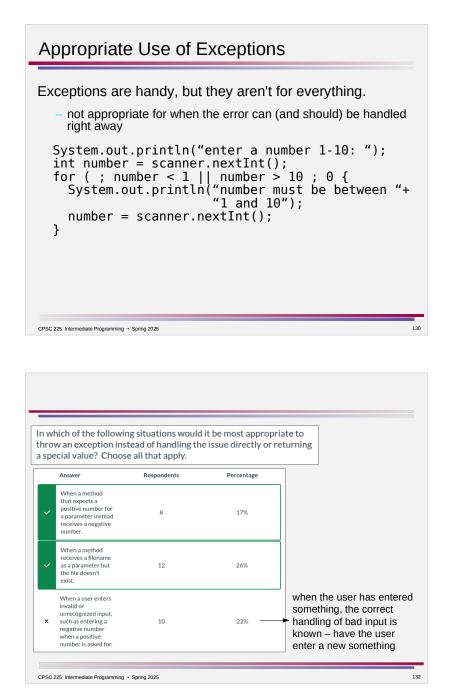
- · determining if the error condition exists is difficult
- errors are signaled with exceptions (but not RuntimeExceptions)

leap before you look because determining if str has a comma requires looking for a comma in str...

int index = str.indexOf(',');
if (index == -1) { ... } // handle error
String before = str.substring(0,index);

```
Example
/**
*
   @exception FileNotFoundException
     if specified file does not exist
   @exception IOException
 *
     if an I/O error occurs while reading the file
 *
 */
public void load ( String filename )
  throws FileNotFoundException, IOException {
  BufferedReader reader =
    new BufferedReader(new FileReader(filename));
  for ( ; true ; ) {
    String line = reader.readLine();
if ( line == null ) { break; }
  }
}
CPSC 225: Intermediate Programming • Spring 2025
                                                             129
```

CPSC 225: Intermediate Programming • Spring 2025



```
int[] numbers = new int[10];
for ( int index = 0 ; index < numbers.length ;</pre>
       index++ ) { numbers[index] = index; }
for ( ; true ; ) {
  String input;
  System.out.print("please type something: ");
  input = scanner.nextLine();
  try {
    int number = Integer.parseInt(input);
    if ( number == 0 ) { break; }
    System.out.println(numbers[number]);
  } catch ( NumberFormatException e ) {
    System.out.println("that wasn't a number!");
  } catch ( ArrayIndexOutOfBoundsException e ) 
    System.out.println("not a legal index!");
         catching a RuntimeException isn't generally the best style
}
         - what should you do instead?
CPSC 225: Intermediate Programming • Spring 2025
                                                        131
```

