Object-Oriented Programming I

Exceptions and the Call Stack

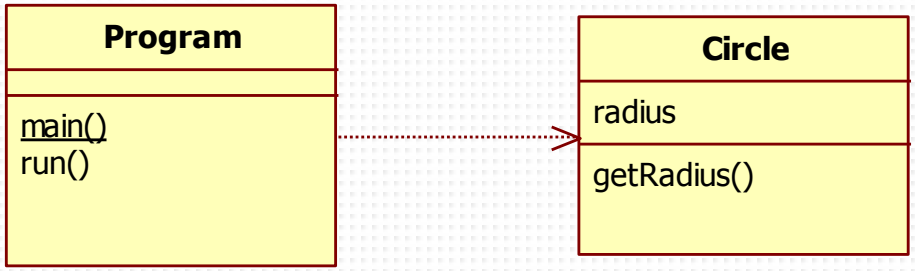
Slides by Magdin Stoica
Updates by Georg Feil

Reading Assignments

- Introduction to Java Programming
  - Sections 5.3, 14.1 to 14.4: Exception Handling and Text I/O
- Head First Java (recommended)
  - Chapter 11: Exception Handling: Risky Behavior
- Java Tutorial: Exceptions (recommended)
  - http://docs.oracle.com/javase/tutorial/essential/exceptions/

Review: Statement Sequence for Method Calls

- Consider a program with two classes "Program" and "Circle"
  - The main() method calls the run() method
  - The run() method creates a Circle object
  - The run() method then calls getRadius()



Statement Sequence for Method Calls

```
1.   package sheridan;
2.
3.   public class Program
4.   {
5.       public static void main(String[] args)
6.       {
7.           statement_1_1;
8.           run();
9.           statement_1_3;
10.      }
11.
12.      private void run()
13.      {
14.          Circle c = new Circle();
15.          double r = c.getRadius();
16.          statement_2_3;
17.      }
18.  }
```

Program.java

```
1.   package sheridan;
2.
3.   public class Circle
4.   {
5.       double radius = 10;
6.
7.       double getRadius()
8.       {
9.           return radius;
10.      }
11.  }
```

Circle.java

Statement Sequence for Method Calls

```
7.  statement_1_1;

8.  run();

9.  statement_1_3;
```

Program.main(...)

```
14. Circle c = new Circle();

15. double r =c.getRadius();

16. statement_2_3;
```

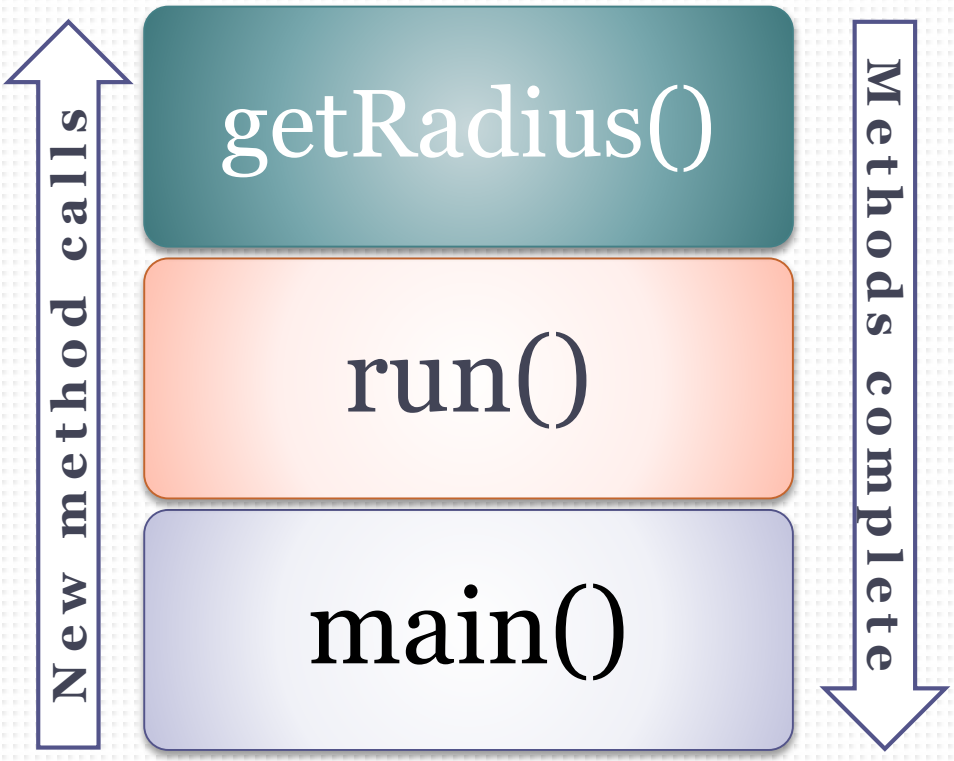
Program.run()

```
9.  return radius;
```

Circle.getRadius()

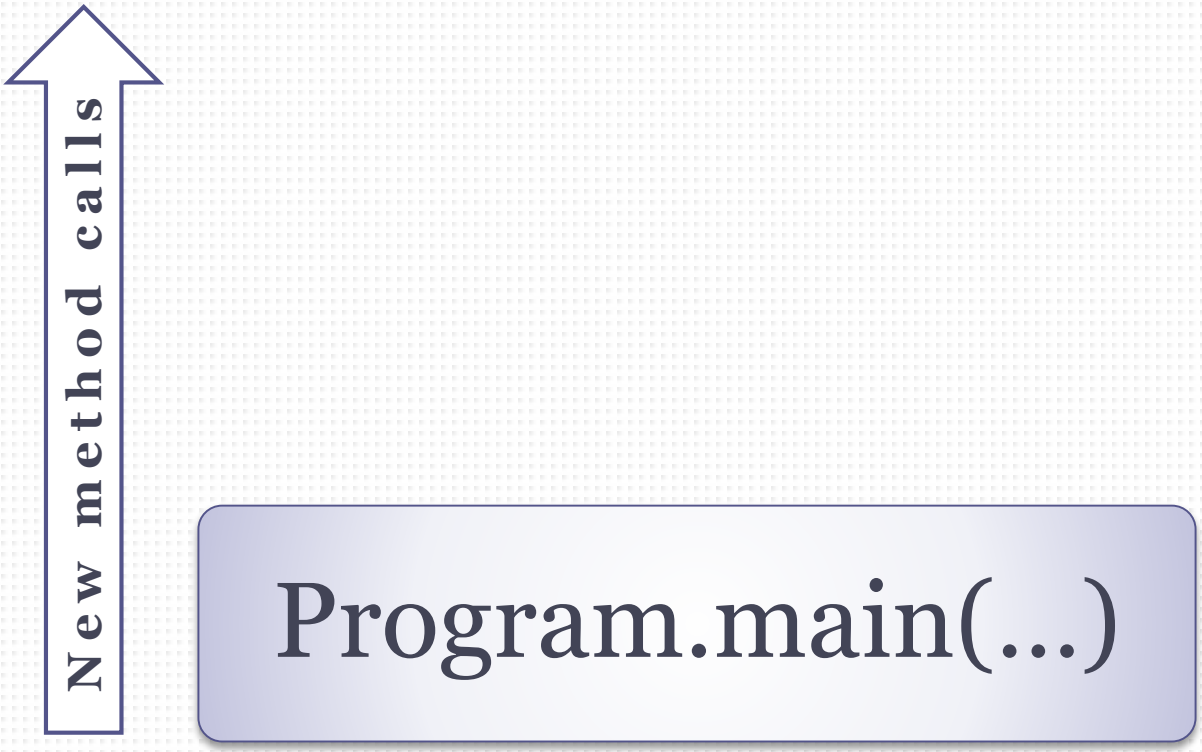
Call Stack

- A method that is executing and has not completed is called an "active method"
- The list of active methods is called the call stack
  - Methods that have been called and have not returned
- Built from the bottom up *like a stack of books* with the bottom-most being the main method
- The height and content of the call stack changes as methods complete their execution



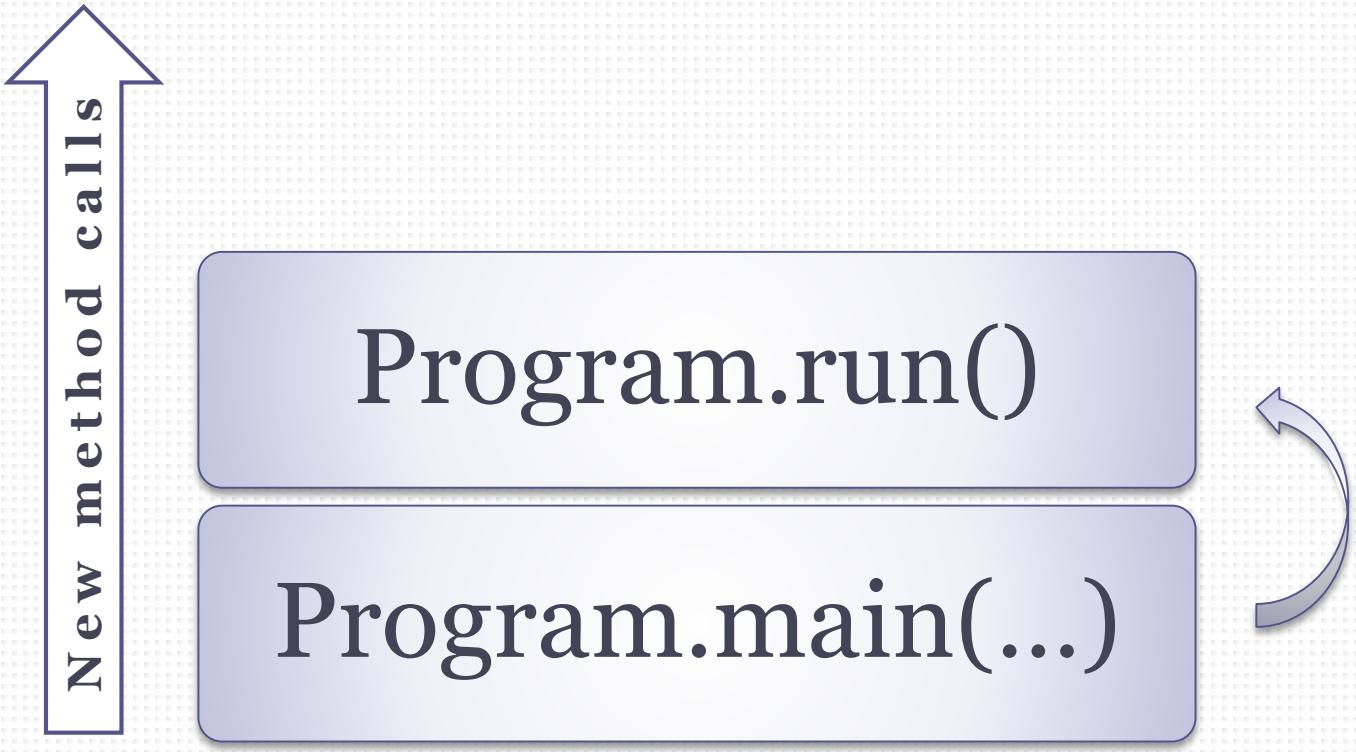
Call Stack

Call Stack and Method Calls



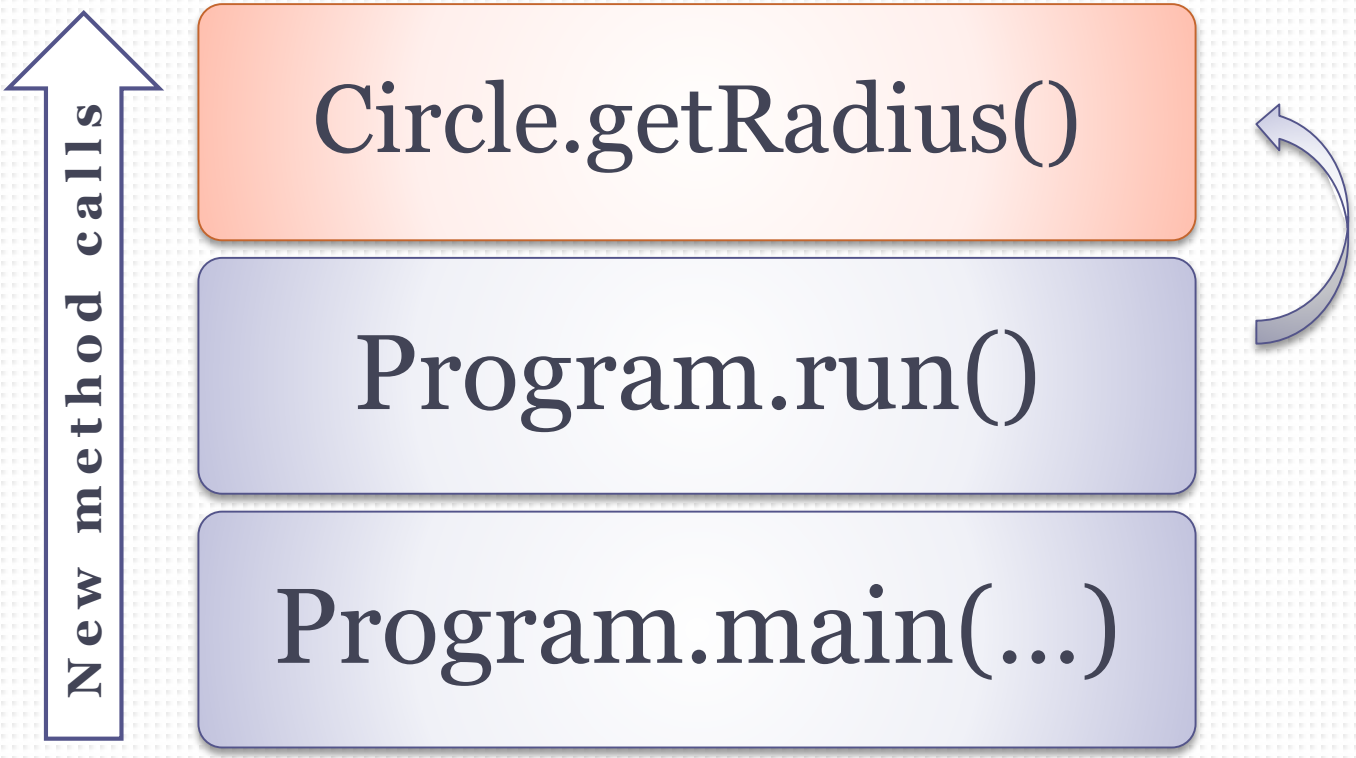
Call stack on line 7 of Program.java

Call Stack and Method Calls



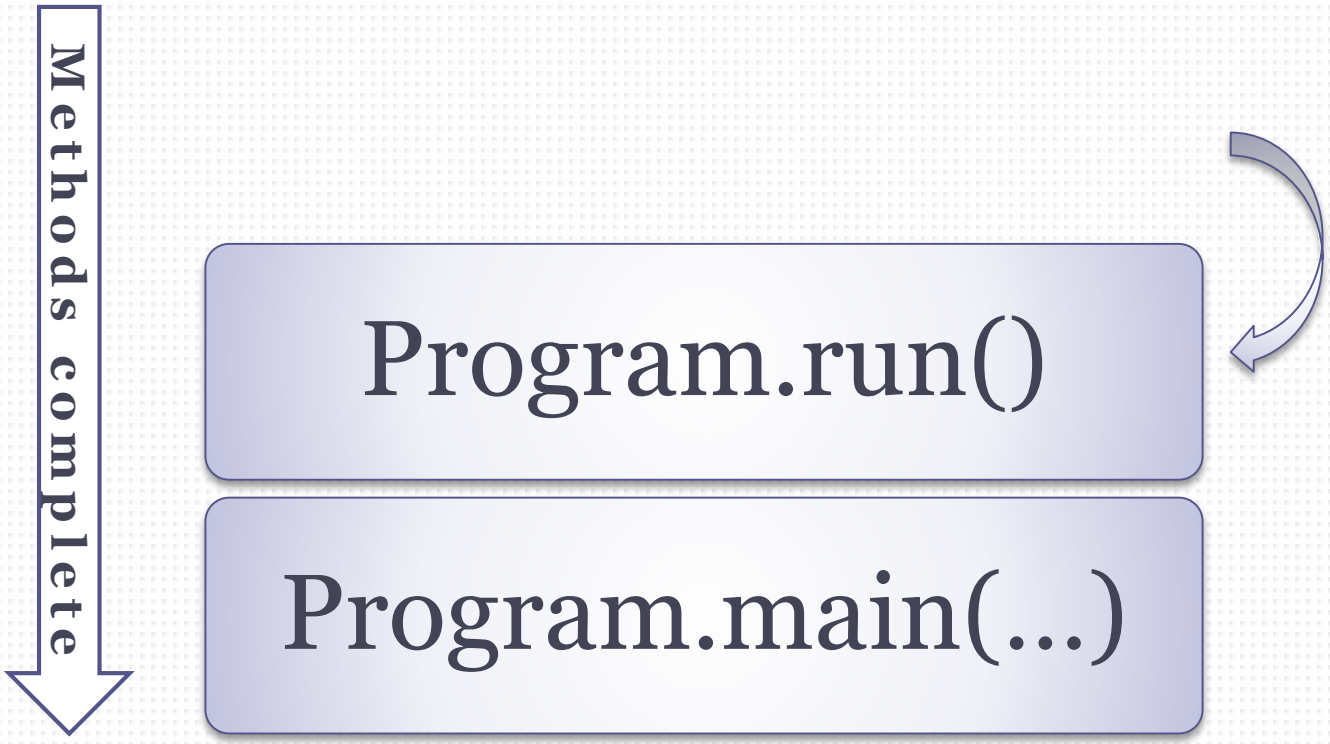
Call stack on line 14 of Program.java

Call Stack and Method Calls



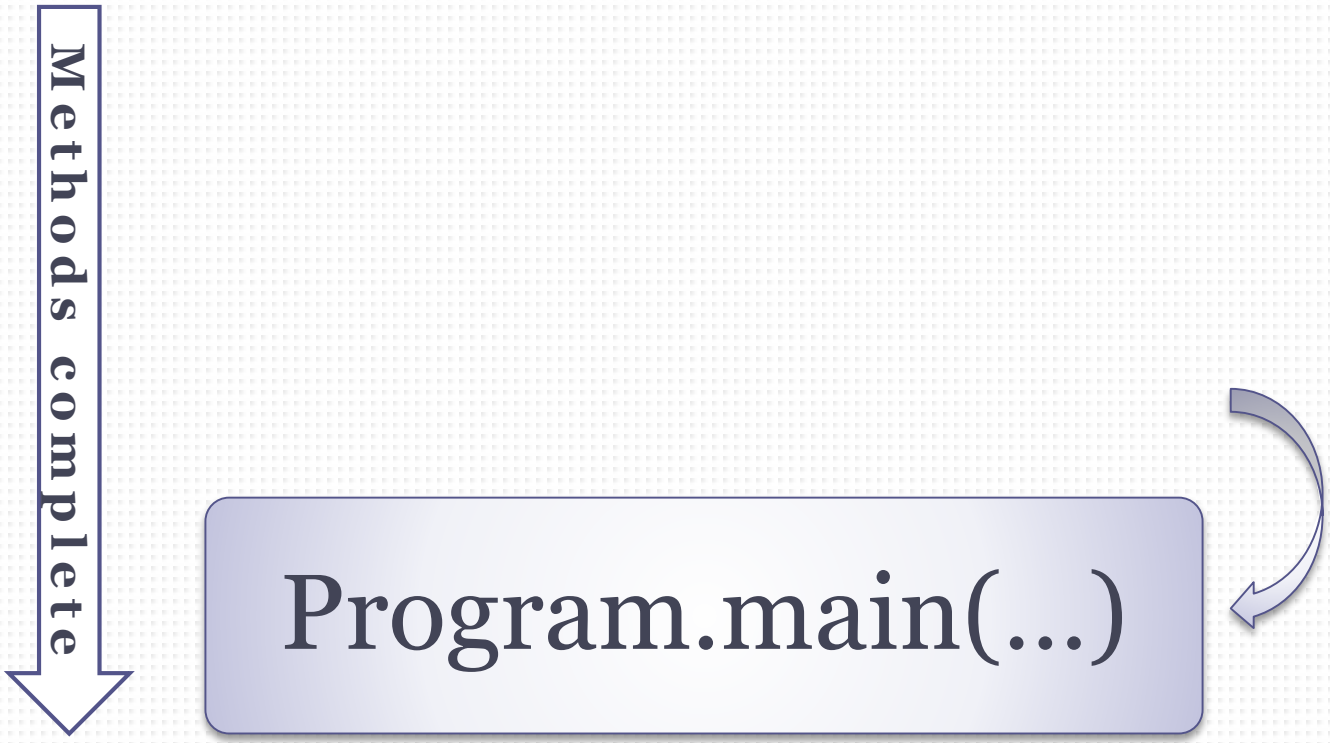
Call stack on line 9 of Circle.java

Call Stack and Method Calls



Call stack on line 16 of Program.java

Call Stack and Method Calls



Call stack on line 9 of Program.java

Java Exceptions

“ try your best and be prepared to catch any errors”

Programming is a risky business

- Many Java library methods can trigger (throw) exceptions
- Your own code can also throw exceptions, on purpose or by accident
  - Code can fail, embrace the errors!
- An exception that is not caught will **crash** your program
  - We've seen this often already, e.g. bad input
- The Java try & catch statements can be used to detect and handle exceptions

Syntax of the try – catch Statement

- All code inside the *try* block is monitored for runtime errors
- This is pseudocode:

```
try {
    statement 1;
    statement 2;
    statement 3;
}
catch (<error object variable declaration>) {
    statement 4;
    statement 5;
}
```

Syntax of the try – catch Statement

- When an exception occurs execution jumps to the *catch* block. It is like a "bucket" which catches errors.

Assume statement 2 generates a runtime error

```
try {
    statement 1;
    statement 2;
    statement 3;
}
catch (Exception e) {
    statement 4;
    statement 5;
}
```

- What do you catch?
  - Objects with error information in them called exceptions. They are instances of a class called Exception

Try-Catch and Program Flow

```
try {
  statement 1;
  statement 2;
  statement 3;
}
catch (Exception e) {
  statement 4;
  statement 5;
}
```

Normal Flow
statement 1;
statement 2;
statement 3;

Exception Flow
statement 1;
statement 2;
statement 4;
statement 5;

Try-Catch and Program Flow

```
try {
  statement 1;
  obj.doSomething();
  statement 3;
}
catch (Exception e) {
  statement 4;
  statement 5;
}
```

Normal Flow

statement 1;

doSomething()
    statement 1;
    ...
    statement N;

statement 3;

Exception Flow

statement 1;

doSomething()
    statement 1;
    statement 2;

statement 4;
statement 5;

Exception Objects

- Things you can do with an exception object
  - Get the error message by calling the getMessage method
  - Print the call stack to find out where the error happened by calling the printStackTrace method

Line 26 will be skipped if the input is incorrect

```
18          try
19          {
20              System.out.print("Please enter the number of rectangle in your drawing
21              Scanner input = new Scanner(System.in);
22              String shapeCountInput = input.nextLine();
23
24              int shapeCount = Integer.parseInt(shapeCountInput);
25
26              System.out.println("The drawing will have " + shapeCount + " rectangles");
27          }
28          catch(Exception e)
29          {
30              System.out.println("The number of rectangle was invalid. Please enter a valid number.\n"
31                                 + e.getMessage());
32          }
```

Exercise 1: Safe User Input

- Write a Java program that
  - Prompts the user to enter a whole number
  - In a method called calculate, displays a random integer between 1 and the number entered
  - Keeps going until the user enters 0 (zero)
- When your program is complete, try this
  - Enter an invalid input value (e.g. blah blah, 8.9)
  - See what happens: identify the runtime error message, call stack and location as shown in the program output
- Add a try-catch block around the input code to catch the error and display a user-friendly message

How much code to put in ‘try’ block?

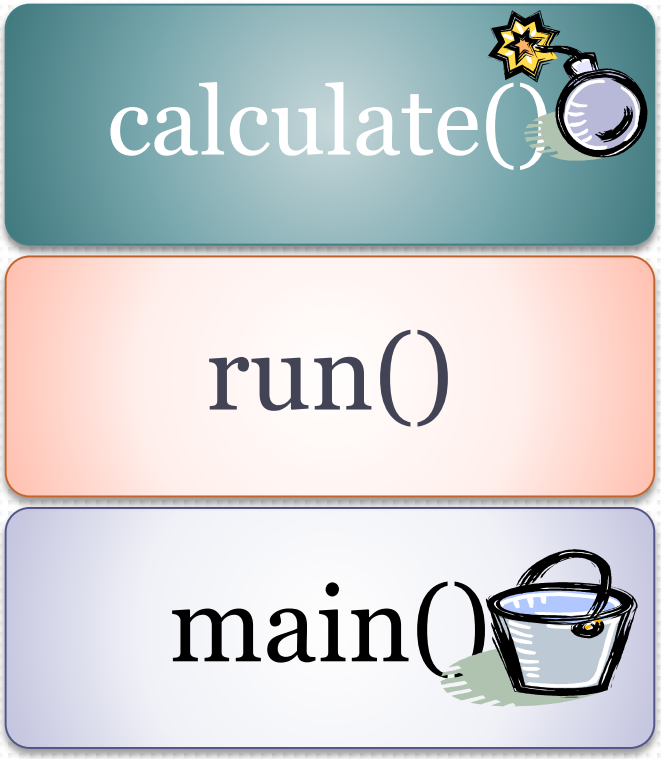
- When you write a ‘try’ block you must decide which lines of your program to put inside
- As a general rule, put in
  - The line(s) that may actually cause a particular exception, or related group of exceptions
  - If needed, a few lines which follow that should definitely be “skipped” if an exception occurs. For example lines which use the input value from Scanner.nextInt.
- Don’t put your whole method, or too many lines, in the try block
  - Start another ‘try’ block later if needed

What should happen in a ‘catch’ block?

- Just printing a message or stack trace is not a very good way to handle an exception
  - You have not recovered from the problem!
  - Users don’t understand stack traces
- Display a user-friendly message
  - Use getMessage, but remember it may not work for some exceptions so don’t rely on it to be the only message
- Recover from the problem, if possible
  - Discard bad user input
  - Use ‘continue’ to go back to the start of a loop, etc…

Exceptions and the Call Stack

- A good thing about exceptions is that they can occur in one method and be caught in another
- main() -> run() -> calculate() -> BadStuffHappenedException
  - If calculate() doesn't have a bucket to catch the exception, run() will get a chance to catch it
  - If run() doesn't have a bucket, main will get a chance to catch it
  - If main () doesn't have a bucket… the JVM will catch the error (crash)
- If your program doesn't catch an exception, the JVM does and it terminates the program



Call Stack

Catching Different Exceptions

- How can you catch multiple types of exceptions?
  - With multiple catch blocks
- A catch block is like a bucket

  - There are smaller (more specific) buckets and…
  - There are bigger (more general) buckets
- You can use as many catch blocks as you want, each for different errors
- Remember to put the smaller buckets first
  - Exception is the biggest bucket so put it last

A try With Multiple Catches

```
try {
  ...
}
catch (IllegalArgumentException e) { // Specific
  ...
}
catch (NumberFormatException e) {  // Specific
  ...
}
catch (Exception e) {   // Any other exception
  ...
}
```

Some Types of Exceptions

- NullPointerException
  - Calling a method on an object that has not been created (is null)
- InputMismatchException
  - Invalid input data while scanning (scanning an integer using Scanner.nextInt() when the input is a not an integer like “abc”)
- NumberFormatException
  - Trying to transform “abc” into a number
- ArrayIndexOutOfBoundsException
  - Accessing array elements from locations that do not exist
- IllegalArgumentException
  - Calling a method with the wrong arguments

Multiple Catches in Java 7

- To reduce code duplication, Java 7 allows you to list several specific exceptions in one catch, separated by '|'
  - Use this if the error handling logic for two or more exceptions is the same

```
try {
  ...
}
catch (IllegalArgumentException |
       NumberFormatException e) {  // Specific
  ...
}
catch (Exception e) {   // Any other exception
  ...
}
```

Triggering Your Own Exceptions

- Throwing exceptions is possible and recommended but only if something is truly wrong
  - It is not replacement for an 'if' statement
  - Exceptions should not appear in "normal" program flow
  - Also called "raising an exception"
- How do you throw an exception?
  - Create an exception object using 'new'
  - Put an error message in it
  - Throw it using the throw keyword:

```
throw new Exception("This code is ready to blow!!!");
```

- When do you throw exceptions?
  - A method is a contract that it does something
  - If a method cannot fulfill an important part of its contract, it may be appropriate to throw an exception

Exercise 2: Throwing Exceptions

- Modify the program from Exercise 1
  - In your calculate() method check if the number entered is less than 1 or greater than 100
  - If so throw an IllegalArgumentException object with the message *"Value out of range. Please use a number between 1 and 100"*
- Add a catch block for IllegalArgumentException
  - In the block print the error message that comes with the exception object using getMessage
- Test your program to see both kinds of exception happen
  - The InputMismatchExeption from ex. 1 and IllegalArgumentException

Other ways to handle errors

- Exceptions are not the only way to handle errors!
- Some other good ways:
  - Return an error code (integer or coded type)
  - Return a special value
    - e.g. Arrays class binarySearch method returns a negative number
- Your use of exceptions should be consistent
  - If you're writing a method of a class in which other methods can throw exceptions, then your method should throw an exception if it encounters a serious problem
  - If other methods return an error code, then your method should return a similar type of error code

Java exception handling summary

- The Java library, or your code, can throw exceptions to indicate a serious problem
- An exception that is not caught will crash your program
- Exceptions are objects containg error info based on the Exception class
- The try and catch statements are used to detect and handle exceptions
- Exceptions are caught by the nearest method on the call stack which has a handler (catch block) for that exception or a more general one
- You can have many catch blocks (put more specific exceptions first)
- To throw an exception use throw followed by an exception object
- Exceptions are not the only way to handle errors