Course Name:

Advanced Java

Lecture 6
Topics to be covered

- Exception Handling (Continued)

Caution

- Remember that, exception subclass must come before any of of their superclasses
- Because, a **catch** statement that uses a superclass will catch exceptions of that type plus any of its subclasses. So, the subclass would never be reached if it come after its superclass
- For example, **ArithmeticException** is a subclass of **Exception**
- Moreover, unreachable code in Java generates error

Example

```
/*  This program contains an error.
 |
   A subclass must come before its superclass in
   a series of catch statements. If not,
   unreachable code will be created and a
   compile-time error will result.
*/
class SuperSubCatch {
  public static void main(String args[]) {
    try {
      int a = 0;
      int b = 42 / a;
    } catch(Exception e) {
      System.out.println("Generic Exception catch.");
    }
    /* This catch is never reached because
       ArithmeticException is a subclass of Exception. */
    catch(ArithmeticException e) { // ERROR - unreachable
      System.out.println("This is never reached.");
    }
  }
}
```

throw

- It is possible for your program to to throw an exception explicitly

  throw *ThrowableInstance*
- Here, *ThrowableInstance* must be an object of type **Throwable** or a subclass **Throwable**
- There are two ways to obtain a **Throwable** objects:
  - Using a parameter into a catch clause
  - Creating one with the **new** operator

Example

```
// Demonstrate throw.
class ThrowDemo {
  static void demoproc() {
    try {
      throw new NullPointerException("demo");
    } catch(NullPointerException e) {
      System.out.println("Caught inside demoproc.");
      throw e; // re-throw the exception
    }
  }

  public static void main(String args[]) {
    try {
      demoproc();
    } catch(NullPointerException e) {
      System.out.println("Recaught: " + e);
    }
  }
}
```

Output:

```
Caught inside demoproc.

Recaught: java.lang.NullPointerException: demo
```

throws

- If a method is capable of causing an exception that it does not handle, it must specify this behavior so that callers of the method can guard themselves against that exception

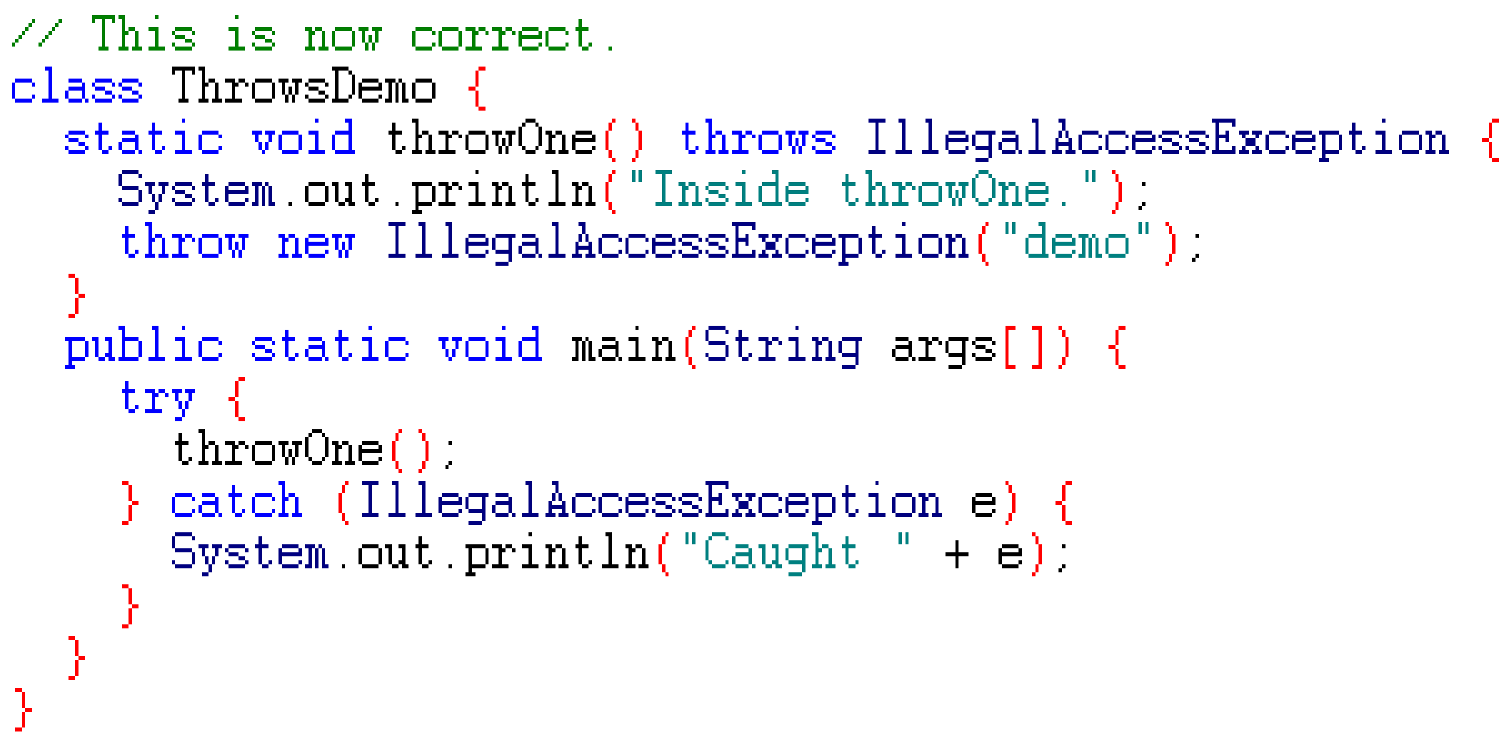
```
type method-name parameter-list) throws exception-list
{
    // body of method
}
```

- It is not applicable for **Error** or **RuntimeException,** or any of their subclasses

Example: incorrect program

```
// This program contains an error and will not compile.
class ThrowsDemo {
  static void throwOne() {
    System.out.println("Inside throwOne.");
    throw new IllegalAccessException("demo");
  }
  public static void main(String args[]) {
    throwOne();
  }
}
```

Example: corrected version



```java
// This is now correct.
class ThrowsDemo {
  static void throwOne() throws IllegalAccessException {
    System.out.println("Inside throwOne.");
    throw new IllegalAccessException("demo");
  }
  public static void main(String args[]) {
    try {
      throwOne();
    } catch (IllegalAccessException e) {
      System.out.println("Caught " + e);
    }
  }
}
```

Output:

```
Inside throwOne.

Caught java.lang.IllegalAccessException: demo
```

finally

- It is used to handle premature execution of a method (i.e. a method open a file upon entry and closes it upon exit)
- **finally** creates a block of code that will be executed after **try/catch** block has completed and before the code following the **try/catch** block
- **finally** clause will execute whether or not an exception is thrown