Throwing Exceptions

v102

```
public class TestCircle
{
  public static void main(String[] args) {

    Circle c = new Circle();

    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a radius: ");

    c.setRadius ( sc.nextInt() );
    System.out.println("Radius: " + c.getRadius() );
  }
}
```

```
public class Circle extends GeometricObject {

  private double radius = 50;

  public double getRadius() {
    return radius;
  }

  public void setRadius(double radius) {
    if (radius <= 0) {
       System.out.println("ERROR! Radius must be > 0.");
    } else {
       this.radius = radius;
    }
  }
```

```
public class TestCircle
{
  public static void main(String[] args) {

    Circle c = new Circle();

    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a radius: ");

    c.setRadius ( sc.nextInt() );
    System.out.println("Radius: " + c.getRadius() );
  }
}
```

```
public class Circle extends GeometricObject {

  private double radius = 50;

  public double getRadius() {
    return radius;
  }

  public void setRadius(double radius) {
    if (radius <= 0) {
       System.out.println("ERROR! Radius must be > 0.");
    } else {
      this.radius = radius;
    }
  }
```

Problem: if -10 is entered for the radius in the red program, then the "**ERROR! Radius must be > 0"** occurs. But the output is displayed to the Terminal window. The program might be a graphics program and the user isn't looking at the terminal window. BETTER TO LET THE RED PROGRAM display the error to the user in its own selected way.

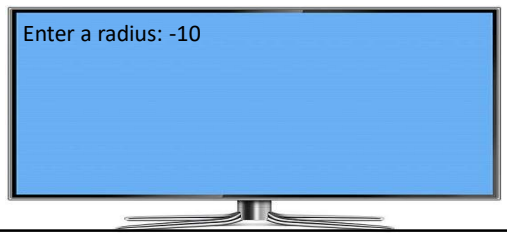
Better to throw and an exception

```
public class TestCircle
{
  public static void main(String[] args) {

    Circle c = new Circle();

    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a radius: ");

    c.setRadius ( sc.nextInt() );
    System.out.println("Radius: " + c.getRadius() );
  }
}
```




```
public class Circle extends GeometricObject {

  private double radius = 50;

  public double getRadius() {
    return radius;
  }


public void setRadius(double radius) {
    if (radius <= 0) {
      IllegalArgumentException e = new IllegalArgumentException();
      throw(e);
      //System.out.println("ERROR! Radius must be > 0.");
    } else {
      this.radius = radius;
    }
  }
```

throw(e) acts like a return statement

Better to throw and an exception

```
public class TestCircle
{
  public static void main(String[] args) {

    Circle c = new Circle();

    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a radius: ");

    c.setRadius ( sc.nextInt() );
    System.out.println("Radius: " + c.getRadius() );
  }
}
```

```
Enter a radius: -10

java.lang.IllegalArgumentException
        at Circle.setRadius(Circle.java:22)
        at TestCircle.main(TestCircle.java:18)
```

```
public class Circle extends GeometricObject {

  private double radius = 50;

  public double getRadius() {
    return radius;
  }

public void setRadius(double radius) {
    if (radius <= 0) {
      IllegalArgumentException e = new IllegalArgumentException();
      throw(e);
      //System.out.println("ERROR! Radius must be > 0.");
    } else {
      this.radius = radius;
    }
  }
```

Sending your own Message with the Exception

```
public class Circle extends GeometricObject {

  private double radius = 50;

  public double getRadius() {
    return radius;
  }


public void setRadius(double radius) {
    if (radius <= 0) {
      IllegalArgumentException e = new IllegalArgumentException(" The radius must be a positive integer.");
      throw(e);
      //System.out.println("ERROR! Radius must be > 0.");
    } else {
      this.radius = radius;
    }
  }


}
```

```
Enter a radius: -10

java.lang.IllegalArgumentException: The radius must be a positive integer.
        at Circle.setRadius(Circle.java:22)
        at TestCircle.main(TestCircle.java:18)
```

Sending your own Message with the Exception

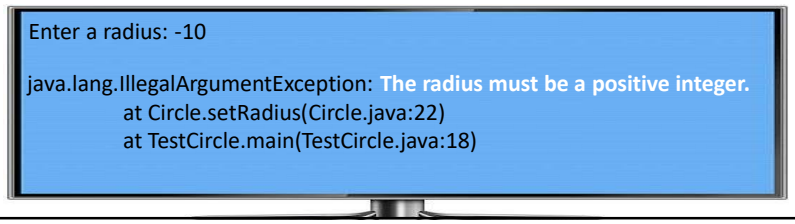
```
public class Circle extends GeometricObject {

   private double radius = 50;

   public double getRadius() {
      return radius;
   }


public void setRadius(double radius) {
      if (radius <= 0) {
         throw new IllegalArgumentException(" The radius must be a positive integer.");
        //System.out.println("ERROR! Radius must be > 0.");
      } else {
         this.radius = radius;
      }
   }


}
```

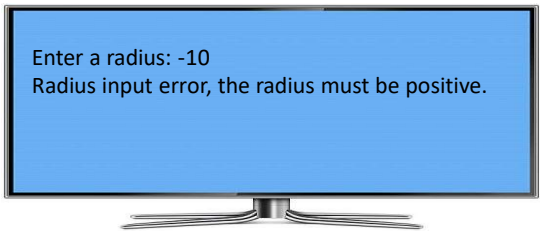



```
public class TestCircle
{
  public static void main(String[] args) {

    Circle c = new Circle();

    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a radius: ");

    try {
      c.setRadius ( sc.nextInt() );
      System.out.println("Radius: " + c.getRadius() );
    }
    catch (IllegalArgumentException e) {
      System.out.println("Radius input error, the radius must be positive.");
    }
  }
}
```



this line doesn't execute, when radius is negative
and
processing continues here

Summary

Throwing an exception is the RIGHT way of telling the **application program** that something went wrong.

It's the responsibility of the **application program** to decide whether to CATCH an unchecked exception or to just let it go.

```
public static void main(String[] args) {

   Circle c = new Circle();

   Scanner sc = new Scanner(System.in);
   System.out.print("Enter a radius: ");

   try {
     c.setRadius ( sc.nextInt() );
     System.out.println("Radius: " + c.getRadius() );
   }
   catch (IllegalArgumentException e) {
      System.out.println("Radius input error, the radius must be positive.");
   }
```

```
public void setRadius(double radius) {
     if (radius <= 0) {
        throw new IllegalArgumentException(" The radius must be a positive integer.");
     } else {
        this.radius = radius;
     }
  }
```

You can make up your own EXCEPTIONS and then throw them.

For example, we could make up our own exception.
For example, we could throw a **BadRadiusException**, an unchecked exception which we have made up.

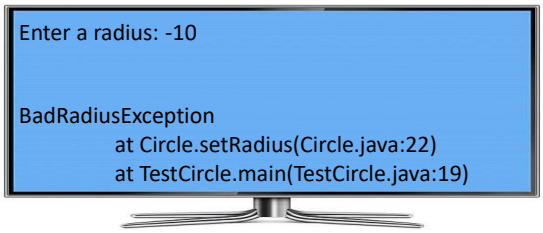
1. Create a new class called, BadRadiusException
2. Extend RuntimeException
3. Add a constructor to it.

```
public void setRadius(double radius) {
     if (radius <= 0) {
        throw new BadRadiusException("The radius must be a positive integer.");
     else {
        this.radius = radius;
     }
  }
```

```
import java.lang.RuntimeException;

public class BadRadiusException extends RuntimeException
{
   public BadRadiusException(String message) {

   }
}
```



You can make up your own EXCEPTIONS and then throw them.

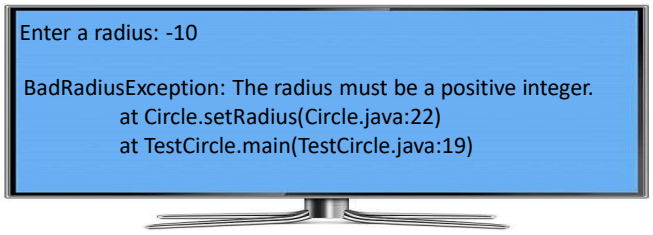
```
public void setRadius(double radius) {
    if (radius <= 0) {
        throw new BadRadiusException("The radius must be a positive integer.");
    else {
        this.radius = radius;
    }
}
```

```
import java.lang.RuntimeException;

public class BadRadiusException extends RuntimeException
{
   //instance var
   private String message;

   public BadRadiusException(String message) {
      this.message = message;
   }

   public String toString() {
      return this.getClass().getSimpleName() + ": " + message;
   }
}
```



QUIZ

See if you can answer these EXAM style questions about

throwing exceptions

1. Write a single line of code that will throw an `IllegalArgumentException` containing the message "Nice try". [1]

1. Write a single line of code that will throw an `IllegalArgumentException` containing the message "Nice try". [1]

```
throw new IllegalArgumentException("Nice try");
```

2. Write code to call a method named `testMethod()`. This method might throw an `IllegalArgumentException`.

Include code to catch that exception and print the message contained in that exception to `System.out`. [4]

2. Write code to call a method named `testMethod()`. This method might throw an `IllegalArgumentException`. Include code to catch that exception and print the message contained in that exception to `System.out`. [4]

```
try {
    testMethod();
} catch ( IllegalArgumentException e ) {
    System.out.println(  e.getMessage()  );
}
```

3. Explain the difference between checked and unchecked exceptions. Give one example of each type of exception. [4]

3. Explain the difference between checked and unchecked exceptions. Give one example of each type of exception. [4]

A checked exception must either be caught or be declared to be thrown by adding the “throws” declaration to the method.

InputMismatchException is unchecked;
FileNotFoundException is checked.

In summary, checked exceptions are required to be either caught or declared, while **unchecked exceptions** can be caught optionally, but they do not have to be declared explicitly in the method signature.

Unchecked exceptions are often used for cases where the failure is beyond the programmer's control (e.g., division by zero)

Checked exceptions are used for conditions that the programmer can anticipate and handle (e.g., file not found).

4. I have a method named `badmethod()`. It takes no arguments and has no return value but when it's called it might throw a special kind of exception called `BadMethodException`.

Write code that will call `badmethod` and print "ERROR" to standard output if it throws the `BadMethodException`. [5]

4. I have a method named `badmethod()`. It takes no arguments and has no return value but when it's called it might throw a special kind of exception called `BadMethodException`.

Write code that will call `badmethod` and print "ERROR" to standard output if it throws the `BadMethodException`. [5]

```
try {
    badmethod();
} catch (BadMethodException e) {
    System.out.println("ERROR");
}
```

5. Depending on how the user behaves, there are 3 exceptions that might be thrown when this code is run. Your job is to figure out which three exceptions might be thrown then re-write the code with additions to it so that if an exception happens, the program prints a user-friendly message and then starts the program again from the beginning. If the user makes it through with no exceptions, the program should end (i.e. not loop back). [14]

```
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);

int[] x = {1,-1,0,2,-2};
int a = sc.nextInt();
int b = sc.nextInt();
int c = x[a];
int d = x[b];
int e = c / d;
System.out.println(e);
}
```

List of Exceptions
ArithmethmeticException
ArrayIndexOutOfBoundsException
FileNotFoundException
InputMismatchException
InterruptedException
NoSuchElementException

5. Depending on how the user behaves, there are 3 exceptions that might be thrown when this code is run. Your job is to figure out which three exceptions might be thrown then re-write the code with additions to it so that if an exception happens, the program prints a user-friendly message and then starts the program again from the beginning. If the user makes it through with no exceptions, the program should end (i.e. not loop back). [14]

```
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);

int[] x = {1,-1,0,2,-2};
int a = sc.nextInt();
int b = sc.nextInt();
int c = x[a];
int d = x[b];
int e = c / d;
System.out.println(e);
}
```

List of Exceptions
ArithmethmeticException
ArrayIndexOutOfBoundsException
FileNotFoundException
InputMismatchException
InterruptedException
NoSuchElementException

```
boolean not_ok = true;
while (not_ok) {
  try {
      // content of main method above goes here
      not_ok = false;
  } catch (ArithmeticException e) {
      System.out.println("Can't divide by 0. Start again.");
  } catch (InputMismatchException e) {
      System.out.println("That wasn't an integer. Start again.");
      sc.next(); // this is not optional
  } catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("Must be in range 0 to 4. Start again.");
  }
}
```