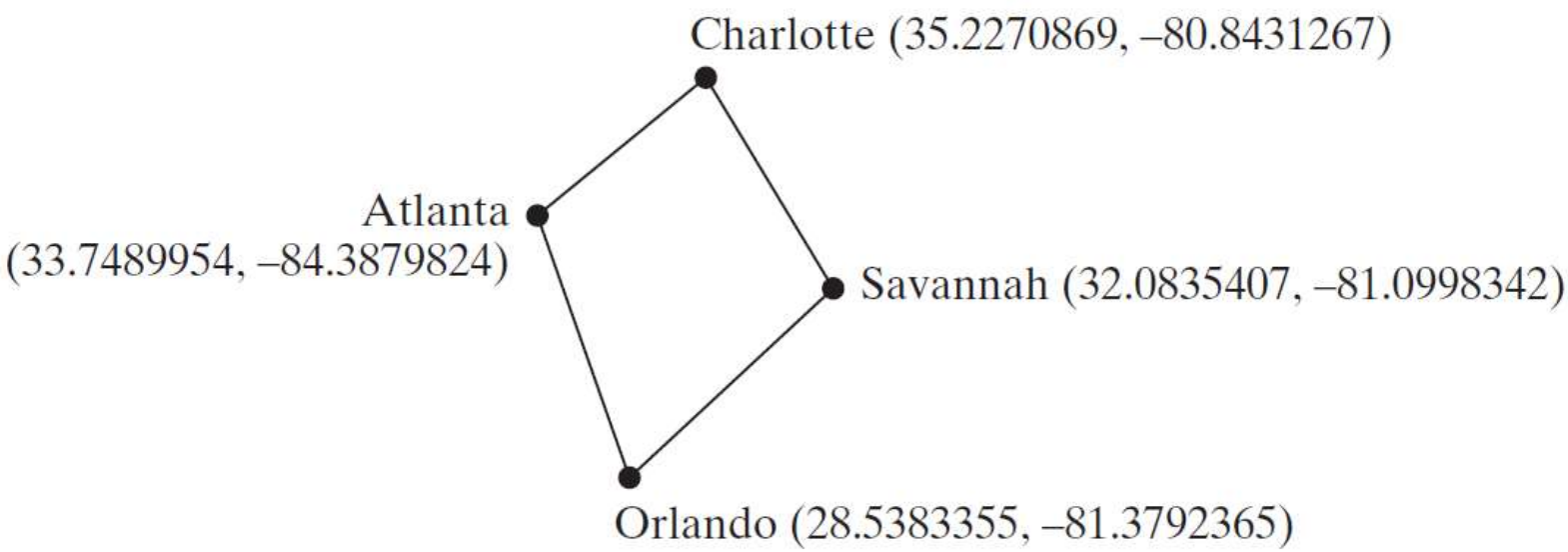
Methods and Exception

Dr. Youna Jung
Northeastern University
yo.jung@northeastern.edu

Motivations

- Suppose you need to estimate the area enclosed by four cities, given the GPS locations (latitude and longitude) of these cities. How would you write a program to solve this problem?



Objectives

- To solve mathematics problems by using the methods in the **Math** class (§4.2).
- To represent characters using the **char** type (§4.3).
- To encode characters using ASCII and Unicode (§4.3.1).
- To represent special characters using the escape sequences (§4.4.2).
- To cast a numeric value to a character and cast a character to an integer (§4.3.3).
- To compare and test characters using the static methods in the **Character** class (§4.3.4).
- To introduce objects and instance methods (**§**4.4).
- To represent strings using the **String** objects (§4.4).
- To return the string length using the **length()** method (§4.4.1).
- To return a character in the string using the **charAt(i)** method (§4.4.2).
- To use the **+** operator to concatenate strings (§4.4.3).
- To read strings from the console (§4.4.4).
- To read a character from the console (§4.4.5).
- To compare strings using the **equals** method and the **compareTo** methods (§4.4.6).
- To obtain substrings (§4.4.7).
- To find a character or a substring in a string using the **indexOf** method (§4.4.8).
- To program using characters and strings (**GuessBirthday**) (§4.5.1).
- To convert a hexadecimal character to a decimal value (**HexDigit2Dec**) (§4.5.2).
- To revise the lottery program using strings (**LotteryUsingStrings**) (§4.5.3).
- To format output using the **System.out.printf** method (§4.6).

The `Math` Class

- Java provides many useful methods in the **`Math`** class for performing common mathematical functions.
  - ✓ Class constants:
    - `PI` and `E`
  - ✓ Class methods:
    1) Trigonometric Methods
    2) Exponent Methods
    3) Service Methods
       - ✓ The rounding, min, max, abs, and random methods

Trigonometric Methods

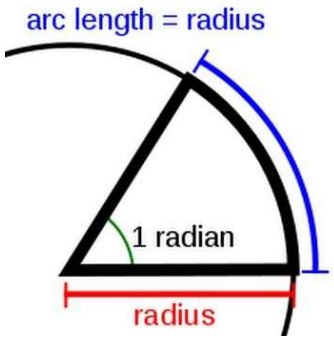
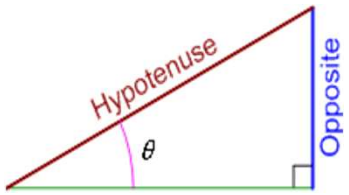
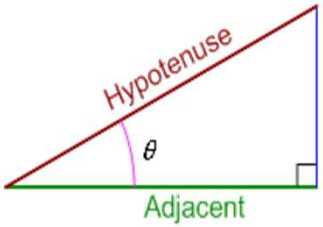


TABLE 4.1 Trigonometric Methods in the Math Class

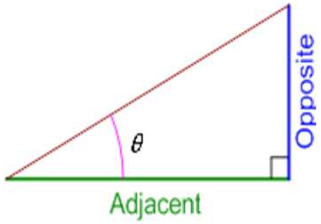
Method	*Description*
`sin(radians)`	Returns the trigonometric sine of an angle in radians.
`cos(radians)`	Returns the trigonometric cosine of an angle in radians.
`tan(radians)`	Returns the trigonometric tangent of an angle in radians.
`toRadians(degree)`	Returns the angle in radians for the angle in degree.
`toDegree(radians)`	Returns the angle in degrees for the angle in radians.
`asin(a)`	Returns the angle in radians for the inverse of sine.
`acos(a)`	Returns the angle in radians for the inverse of cosine.
`atan(a)`	Returns the angle in radians for the inverse of tangent.



$$\text{Sin}\,\theta = \frac{\text{Opposite}}{\text{Hypotenuse}}$$



$$\text{Cos}\,\theta = \frac{\text{Adjacent}}{\text{Hypotenuse}}$$



$$\text{Tan}\,\theta = \frac{\text{Opposite}}{\text{Adjacent}}$$

Trigonometric Methods

- Examples

```
Math.toDegrees(Math.PI / 2) returns 90.0
Math.toRadians(30) returns 0.5236 (same as π/6)
Math.sin(0) returns 0.0
Math.sin(Math.toRadians(270)) returns -1.0
Math.sin(Math.PI / 6) returns 0.5
Math.sin(Math.PI / 2) returns 1.0
Math.cos(0) returns 1.0
Math.cos(Math.PI / 6) returns 0.866
Math.cos(Math.PI / 2) returns 0
Math.asin(0.5) returns 0.523598333 (same as π/6)
Math.acos(0.5) returns 1.0472 (same as π/3)
Math.atan(1.0) returns 0.785398 (same as π/4)
```

Exponent Methods

TABLE 4.2 Exponent Methods in the Math Class

Method	*Description*
`exp(x)`	Returns e raised to power of x (e^x).
`log(x)`	Returns the natural logarithm of x ($\ln(x) = \log_e(x)$).
`log10(x)`	Returns the base 10 logarithm of x ($\log_{10}(x)$).
`pow(a, b)`	Returns a raised to the power of b (a^b).
`sqrt(x)`	Returns the square root of x ($\sqrt{x}$) for x >= 0.

Exponent Methods

- Examples

```
Math.exp(1) returns 2.71828
Math.log(Math.E) returns 1.0
Math.log10(10) returns 1.0
Math.pow(2, 3) returns 8.0
Math.pow(3, 2) returns 9.0
Math.pow(4.5, 2.5) returns 22.91765
Math.sqrt(4) returns 2.0
Math.sqrt(10.5) returns 4.24
```

Rounding Methods

TABLE 4.3 Rounding Methods in the Math Class

Method	*Description*
`ceil(x)`	x is rounded up to its nearest integer. This integer is returned as a double value.
`floor(x)`	x is rounded down to its nearest integer. This integer is returned as a double value.
`rint(x)`	x is rounded up to its nearest integer. If x is equally close to two integers, the even one is returned as a double value.
`round(x)`	Returns (int)Math.floor(x + 0.5) if x is a float and returns (long)Math.floor(x + 0.5) if x is a double.

Rounding Methods

- Examples

```
Math.ceil(2.1) returns 4.0
Math.ceil(2.0) returns 2.0
Math.ceil(-2.0) returns -2.0
Math.ceil(-2.1) returns -2.0
Math.floor(2.1) returns 2.0
Math.floor(2.0) returns 2.0
Math.floor(-2.0) returns -2.0
Math.floor(-2.1) returns -4.0
Math.rint(2.1) returns 2.0
Math.rint(-2.0) returns -2.0
Math.rint(-2.1) returns -2.0
Math.rint(2.5) returns 2.0
Math.rint(4.5) returns 4.0
Math.rint(-2.5) returns -2.0
Math.round(2.6f) returns 3 // Returns int
Math.round(2.0) returns 2 // Returns long
Math.round(-2.0f) returns -2 // Returns int
Math.round(-2.6) returns -3 // Returns long
Math.round(-2.4) returns -2 // Returns long
```

Rounding Methods

- Correction

Test.java

```java
public class Test {
  public static void main(String[] args){
      System.out.println(Math.ceil(2.1));
      System.out.println(Math.ceil(2.0));
      System.out.println(Math.ceil(-2.0));
      System.out.println(Math.ceil(-2.1));
      System.out.println(Math.floor(2.1));
      System.out.println(Math.floor(2.0));
      System.out.println(Math.floor(-2.0));
      System.out.println(Math.floor(-2.1));
      System.out.println(Math.rint(2.1));
      System.out.println(Math.rint(-2.0));
      System.out.println(Math.rint(-2.1));
      System.out.println(Math.rint(2.5));
      System.out.println(Math.rint(4.5));
      System.out.println(Math.rint(-2.5));
      System.out.println(Math.round(2.6f));
      System.out.println(Math.round(2.0));
      System.out.println(Math.round(-2.0f));
      System.out.println(Math.round(-2.6));
      System.out.println(Math.round(-2.4));
  }
}
```

Problems @ Javac

```
<terminated> Test [Java
3.0
2.0
-2.0
-2.0
2.0
2.0
-2.0
-3.0
2.0
-2.0
-2.0
2.0
4.0
-2.0
3
2
-2
-3
-2
```

min(), max(), and abs()

- **`max(a, b)`** and **`min(a, b)`**
  - ✓ Returns the maximum or minimum of two parameters.
- **`abs(a)`**
  - ✓ Returns the absolute value of the parameter.
- **`random()`**
  - ✓ Returns a random `double` value in the range [0.0, 1.0).

```
Math.max(2, 3)
Math.max(2.5, 3)
Math.min(2.5, 3.6)
Math.abs(-2)
Math.abs(-2.1)
```

```
3
3.0
2.5
2
2.1
```

min(), max(), and abs()

- Correction

Test.java

```
1
2  public class Test {
3      public static void main(String[] args){
4          System.out.println(Math.max(2,3));
5          System.out.println(Math.max(2.5,3));
6          System.out.println(Math.min(2.5, 4.6));
7          System.out.println(Math.abs(-2));
8          System.out.println(Math.abs(-2.1));
9      }
10 }
11
```

Problems @ J

```
<terminated> Test [.
3
3.0
2.5
2
2.1
```

The random Method

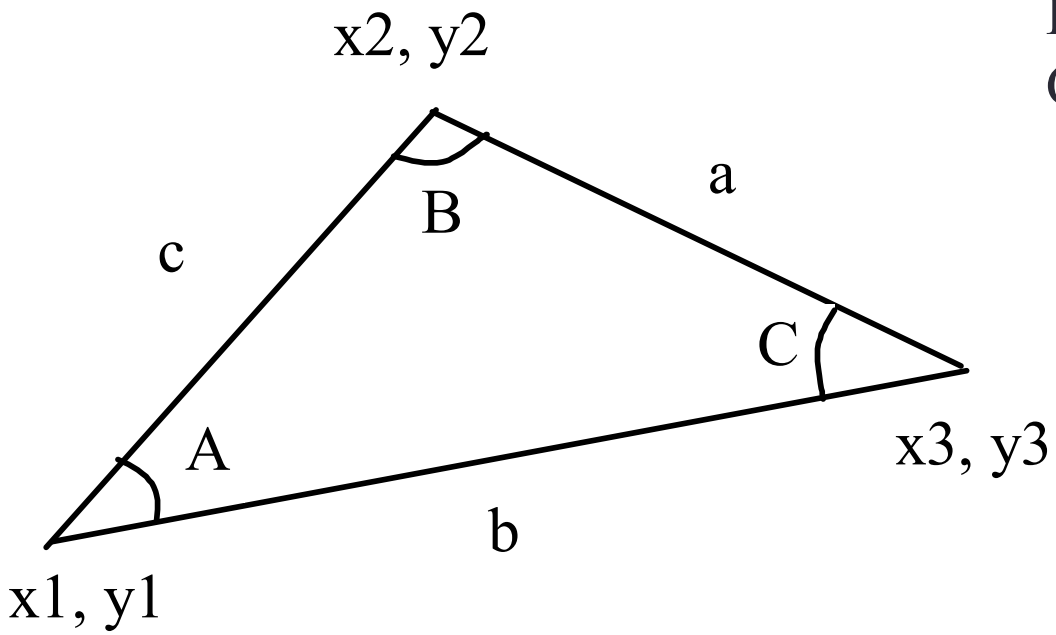
- Generates a random double value greater than or equal to 0.0 and less than 1.0 (*0.0 ≤ Math.random()< 1.0*).
- In general,

```
a + Math.random() * b
```

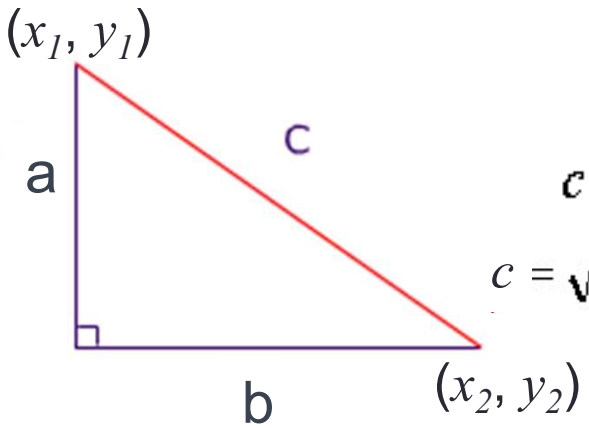
  - ✓ returns a random number between ***a*** and ***a + b***, excluding *a + b*.
    - ex) `(int)(Math.random() * 10)` → *Returns a random integer between 0 and 9.*
    - ex) `50 + (int)(Math.random() * 50)` → *Returns a random integer between 50 and 99.*

Computing Angles of a Triangle

- Write a program that prompts the user to enter the x- and y-coordinates of the three corner points in a triangle and then displays the triangle's angles.



A = acos((a * a - b * b - c * c) / (-2 * b * c))
B = acos((b * b - a * a - c * c) / (-2 * a * c))
C = acos((c * c - b * b - a * a) / (-2 * a * b))



$a^2+b^2 = c^2$

$$c = \sqrt{a^2 + b^2}$$

$$c = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

A = acos((a * a - b * b - c * c) / (-2 * b * c))
B = acos((b * b - a * a - c * c) / (-2 * a * c))
C = acos((c * c - b * b - a * a) / (-2 * a * b))

```
c:\book>java ComputeAngles
Enter three points: 1.0
1.0
6.5
1.0
6.5
2.5
The three angles are 15.26 90.0 74.74
```

```java
import java.util.Scanner;
public class ComputeAngles {
  public static void main(String[] args) {

    Scanner input = new Scanner(System.in);
    System.out.print("Enter three points: ");
    double x1 = input.nextDouble();
    double y1 = input.nextDouble();
    double x2 = input.nextDouble();
    double y2 = input.nextDouble();
    double x3 = input.nextDouble();
    double y3 = input.nextDouble();

    double a = Math.sqrt((x2 - x3) * (x2 - x3) + (y2 - y3) * (y2 - y3));
    double b = Math.sqrt((x1 - x3) * (x1 - x3) + (y1 - y3) * (y1 - y3));
    double c = Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));

    double A = Math.toDegrees(Math.acos((a*a - b*b - c*c) / (-2*b*c)));
    double B = Math.toDegrees(Math.acos((b*b - a*a - c*c) / (-2*a*c)));
    double C = Math.toDegrees(Math.acos((c*c - b*b - a*a) / (-2*a*b)));

    System.out.println("The three angles are " + Math.round(A * 100) / 100.0
     + " " + Math.round(B * 100) / 100.0 + " " + Math.round(C*100) / 100.0);
  }
}
```

ComputeAngles

Character Data Type

- A character data type (**`char`**) represents a single character
- A character literal is enclosed in single quotation marks(‘)
  - ✓ ex) `char letter = 'A';` (ASCII)
  - ✓ ex) `char numChar = '4';` (ASCII)
  - ✓ ex) `char letter = '\u0041';` (Unicode)
  - ✓ ex) `char numChar = '\u0034';` (Unicode)
- NOTE: The increment and decrement operators can also be used on `char` variables to get the next or preceding Unicode character.
  - ✓ ex) `char ch = 'a';` // The decimal value of ‘a’ is 97
  - ✓ ex) `System.out.println(++ch);` `b`

Unicode Format

- Java characters use **Unicode**, a 16-bit encoding scheme established by the Unicode Consortium to support the interchange, processing, and display of written texts in the world's diverse languages.
  - ✓ Unicode takes 2 bytes,
  - ✓ preceded by \u,
  - ✓ expressed in 4 hexadecimal numbers that run from '\u0000' to '\uFFFF'.
    - To represent 1 hexadecimal number, 4bits ($16=2^4$) are required
    - So, Unicode can represent 65536 ($16^4= 65536$) characters.

ASCII Code

- An 8-bit encoding scheme for representing all *uppercase* and *lowercase letters*, *digits*, *punctuation marks*, and *control characters*

TABLE 4.4 ASCII Code for Commonly Used Characters

Characters	*Code Value in Decimal*	*Unicode Value*
'0' to '9'	48 to 57	\u0030 to \u0039
'A' to 'Z'	65 to 90	\u0041 to \u005A
'a' to 'z'	97 to 122	\u0061 to \u007A

- Unicode includes ASCII code
  - With \u0000 to \u007F
  - You can use ASCII characters such as 'a' as well as Unicode in a Java program
    - `char letter = 'a';` is equivalent to `char letter = '\u0061';`

Appendix B: ASCII Code

- ASCII Character Set is a subset of the Unicode from \u0000 to \u007f

TABLE B.2 ASCII Character Set in the Hexadecimal Index

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht	nl	vt	ff	cr	so	si
1	dle	dcl	dc2	dc3	dc4	nak	syn	etb	can	em	sub	esc	fs	gs	rs	us
2	sp	!	“	#	$	%	&	’	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	del

ASCII Code

ASCII Character Set in the Decimal Index

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	nl	vt	ff	cr	so	si	dle	dcl	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	″	#	$	%	&	’
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	‘	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	del		

Escape Sequences for Special Characters

- Java uses a special notation to represent special characters (***escape sequence***)
  - ✓ consists of a backslash(\) followed by a character or a combination of digits

```
System.out.prinln("He said "java is fun"");
System.out.prinln("He said \"java is fun\"");
```

Error!

OK

Escape Sequence	*Name*	*Unicode Code*	*Decimal Value*
\b	Backspace	\u0008	8
\t	Tab	\u0009	9
\n	Linefeed	\u000A	10
\f	Formfeed	\u000C	12
\r	Carriage Return	\u000D	13
\\	Backslash	\u005C	92
\"	Double Quote	\u0022	34

Casting between `char` and Numeric Types

- A `char` can be cast into any numeric type, and vice versa.
- Cast numeric type into `char`
  - ✓ When a number is cast into a `char`, only its lower 16 bits of data are used.

```
Char ch = (char) 0EAB0041; //The lower 16 bits hexcode 0041
System.out.prinln(ch);      A
```

  - ✓ When a floating-point value is cast into a `char`, the floating-point value is first cast into an **`int`**, which is then cast into a **`char`**

```
Char ch = (char) 65.25; //Decimal 65 is assigned to
System.out.prinln(ch);  A
```

Casting between `char` and Numeric Types

- Implicit casting can be used if the result of a casting fits into the target variable. Otherwise, explicit casting MUST be used.
  - ✓ Any positive integer between **0** and **FFFF** in hexadecimal can be cast into a character implicitly. However, any number not in this range MUST be cast into a char explicitly.

```
byte b = 'a';                  OK
int i = 'a';                   OK
byte b = '\uFFF4F';            Error!
byte b = (byte)'\uFFF4';       OK
```

Casting between char and Numeric Types

- All numeric operators can be applied to `char` operands
  - ✓ if an operand is a number or a character
    - → A `char` operand is automatically cast into a number

```
int i = '2' + '3'; // (int)'2' is 50 and (int)'3' is 51
System.out.println("i is " + i); // i is 101
int j = 2 + 'a'; // (int)'a' is 97
System.out.println("j is " + j); // j is 99
System.out.println(j + " is the Unicode for character "
  + (char)j); // 99 is the Unicode for character c
System.out.println("Chapter " + '2');
```

```
i is 101
j is 99
99 is the Unicode for character c
Chapter 2
```

Comparing Characters

- Two characters can be compared using the relational operators
  - ✓ This is done by comparing the Unicodes of the two characters

`'a' < 'b'` is true because the Unicode for `'a'` (97) is less than the Unicode for `'b'` (98).
`'a' < 'A'` is false because the Unicode for `'a'` (97) is greater than the Unicode for `'A'` (65).
`'1' < '8'` is true because the Unicode for `'1'` (49) is less than the Unicode for `'8'` (56).

Testing Characters

- To test whether a character is a *number*, a *letter*, an *uppercase letter*, or a *lowercase letter*.
  - ✓ The Unicodes for lowercase letters are consecutive integers starting from the Unicode for 'a'

```
if (ch >= 'A' && ch <= 'Z')
  System.out.println(ch + " is an uppercase letter");
else if (ch >= 'a' && ch <= 'z')
  System.out.println(ch + " is a lowercase letter");
else if (ch >= '0' && ch <= '9')
  System.out.println(ch + " is a numeric character");
```

Methods in the `Character` Class

- For convenience, Java provides the following methods in the `Character` class.

TABLE 4.6 Methods in the Character Class

Method	*Description*
`isDigit(ch)`	Returns true if the specified character is a digit.
`isLetter(ch)`	Returns true if the specified character is a letter.
`isLetterOrDigit(ch)`	Returns true if the specified character is a letter or digit.
`isLowerCase(ch)`	Returns true if the specified character is a lowercase letter.
`isUpperCase(ch)`	Returns true if the specified character is an uppercase letter.
`toLowerCase(ch)`	Returns the lowercase of the specified character.
`toUpperCase(ch)`	Returns the uppercase of the specified character.

Methods in the `Character` Class

- Examples

```
System.out.println("isDigit('a') is " + Character.isDigit('a'));
System.out.println("isLetter('a') is " + Character.isLetter('a'));
System.out.println("isLowerCase('a') is "
  + Character.isLowerCase('a'));
System.out.println("isUpperCase('a') is "
  + Character.isUpperCase('a'));
System.out.println("toLowerCase('T') is "
  + Character.toLowerCase('T'));
System.out.println("toUpperCase('q') is "
  + Character.toUpperCase('q'));
```

```
isDigit('a') is false
isLetter('a') is true
isLowerCase('a') is true
isUpperCase('a') is false
toLowerCase('T') is t
toUpperCase('q') is Q
```

The `String` Type

- A string is a sequence of characters.
  - ✓ The `char` type only represents one character.
  - ✓ To represent a string of characters, use the data type called **`String`**.

- **`String`** is actually a predefined class in the Java library just like the **`Scanner`** class.
  - ✓ The **`String`** type is NOT a primitive type. It is known as a *reference type*.

```
String message = "Welcome to Java";
```

Methods for `String` Class

TABLE 4.7 Simple Methods for `String` Objects

Method	*Description*
`length()`	Returns the number of characters in this string.
`charAt(index)`	Returns the character at the specified index from this string.
`concat(s1)`	Returns a new string that concatenates this string with string s1.
`toUpperCase()`	Returns a new string with all letters in uppercase.
`toLowerCase()`	Returns a new string with all letters in lowercase
`trim()`	Returns a new string with whitespace characters trimmed on both sides.

Methods for `String` Objects

- 2 Types of Methods
  - ✓ Instance methods
    - The methods in the preceding table can only be invoked from a specific instance.
  - ✓ Static method (non-instance method)
    - A static method can be invoked WITHOUT using an object.
      - e.g.) All the methods defined in the **`Math`** class are static methods.

- `String`s are objects in Java.
  - ✓ Syntax to invoke an instance method

```
referenceVariable.methodName(arguments);
```

Getting String Length

- **`length()`** method
  - ✓ Return the number of characters in a string
    - Example

```
String message = “Welcome to Java”;
System.out.println("The length of " + message + "is " + message.length());
```

```
The length of Welcome to Java is 15
```

  - ✓ NOTE
    - You can use the string literal to refer directly to strings without creating new variables

```
“Welcome to Java”.length()
```

```
15
```

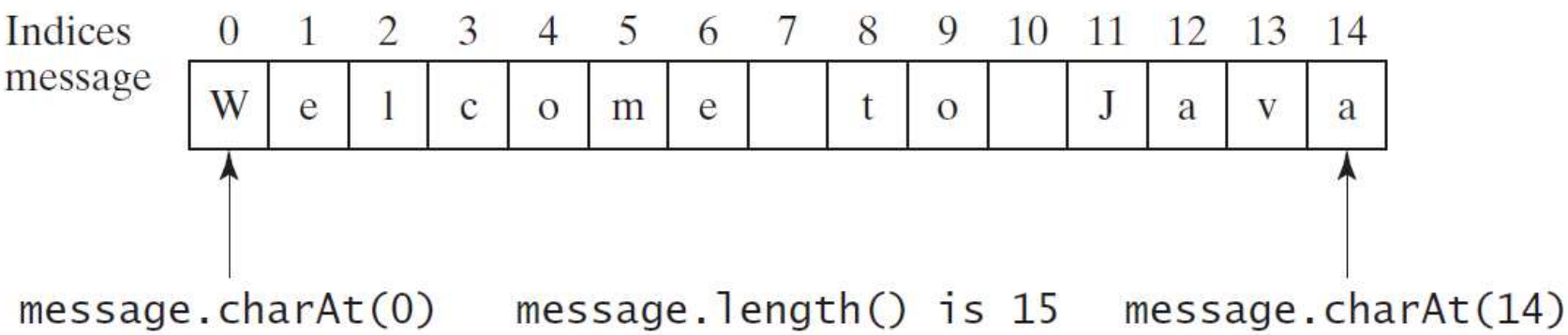
    - “” is an *empty string*.

```
“”.length()
```

```
0
```

Getting Characters from a String

- The **`s.charAt(index)`** method can be used to retrieve a specific character in a string `s`, where the index is between **`0`** and **`s.length()-1`**.

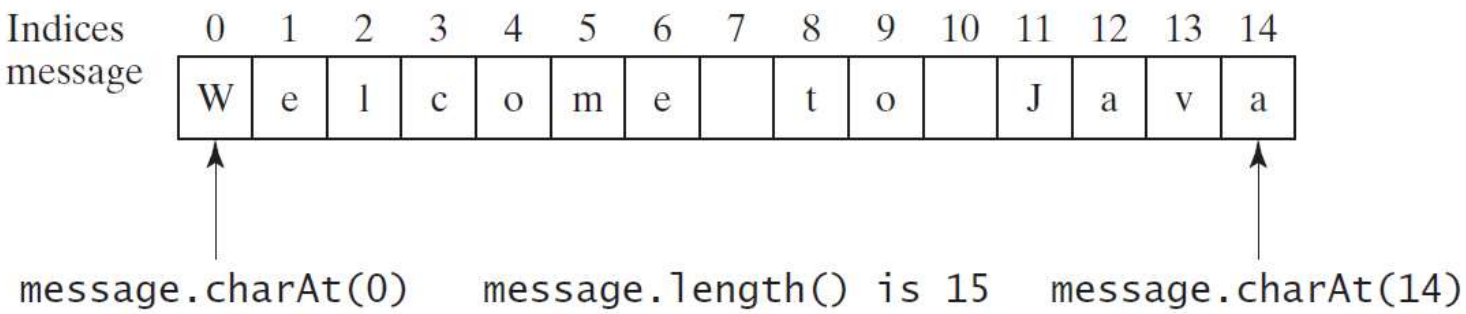



```
String message = "Welcome to Java";
System.out.println("The first character in message is " + message.charAt(0));
```

The first character in message is W

CAUTION

- Attempting to access characters in a string *s* out of bounds is a common error.
  - ✓ Do not use an index beyond `s.length()-1`.




```
String message = "Welcome to Java";
System.out.println(message.charAt(message.length()));
```

StringIndexOutOfBoundsException!

String Concatenation

- Two ways to concatenate two strings
  1) Use the plus (+) operator

```
String s3 = s1 + s2;
```

  2) The `concat()` method

```
String s3 = s1.concat(s2);
```

- Examples

The number is converted into a string and then concatenated.

```
String message = "Welcome " + "to " + "Java";
String s = "Chapter" + 2; // s becomes Chapter2
String s1 = "Supplement" + 'B'; // s1 becomes SupplementB
```

```
Welcome to Java
Chapter2
SupplementB
```

String Concatenation

- The augmented += operator can also be used for string concatenation.
  - ✓ Example

```
String message = " Welcome to Java";
message += " and Java is fun";
System.out.println(message);
```

```
Welcome to Java and Java is fun
```

Converting Strings

- **`toLowerCase()`**
  - ✓ Return a new string with all letters in lowercase
    - ex) `"Welcome".toLowerCase()` → welcome
- **`toUpperCase()`**
  - ✓ Return a new string with all letters in uppercase
    - ex) `"Welcome".toUpperCase()` → WELCOME
- **`trim()`**
  - ✓ Return a new string with whitespace characters trimmed on both sides
    - Whitespace characters – ‘’, \t, \f, \r, or \n
    - ex) `"  Welcome  ".trim()` → "Welcome"

Reading a String from the Console

- Invoke the **`next()`** method or the **`nextLine()`** method on a `Scanner` object.
  1) **`next()`**
     - Reads a **string** that ends with a **whitespace character**

```
Scanner input = new Scanner(System.in);
System.out.print("Enter three words separated by spaces: ");
String s1 = input.next();
String s2 = input.next();
String s3 = input.next();
System.out.println("s1 is " + s1);
System.out.println("s2 is " + s2);
System.out.println("s3 is " + s3);
```

```
Enter three words separated by spaces: Welcome to Java ↵Enter
s1 is Welcome
s2 is to
s3 is Java
```

Reading a String from the Console

2) **nextLine()**

– Read an **Entire Line** of text that ends with **the Enter key** pressed.

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a line: ");
String s = input.nextLine();
System.out.println("The line entered is " + s);
```

```
Enter a line: Welcome to Java ↵Enter
The line entered is Welcome to Java
```

Reading a Character from the Console

- To read a character from the console
  1) Use the **`nextLine()`** method to read a string
  2) Invoke the **`charAt(0)`** method on the string to return a character

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a character: ");
String s = input.nextLine();
char ch = s.charAt(0);
System.out.println("The character entered is " + ch);
```

Comparing Strings

TABLE 4.8 Comparison Methods for `String` Objects

Method	*Description*
`equals(s1)`	Returns true if this string is equal to string `s1`.
`equalsIgnoreCase(s1)`	Returns true if this string is equal to string `s1`; it is case insensitive.
`compareTo(s1)`	Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than `s1`.
`compareToIgnoreCase(s1)`	Same as `compareTo` except that the comparison is case insensitive.
`startsWith(prefix)`	Returns true if this string starts with the specified prefix.
`endsWith(suffix)`	Returns true if this string ends with the specified suffix.
`contains(s1)`	Returns true if s1 is a substring in this string.

Comparing Strings

- 3 ways to compare two strings
  1) Use the **==** operator
     - checks only whether `s1` and `s2` **refer to the same object**
     - does NOT check if they have the same contents

```
if (string1 == string2)
  System.out.println("string1 and string2 are the same object");
else
  System.out.println("string1 and string2 are different objects");
```

Comparing Strings

2) Use the **`equals()`** method
   - checks whether `s1` and `s2` have **the same content**.

```
if (string1.equals(string2))
  System.out.println("string1 and string2 have the same contents");
else
  System.out.println("string1 and string2 are not equal");
```

```
String s1 = "Welcome to Java";
String s2 = "Welcome to Java";
String s3 = "Welcome to C++";
System.out.println(s1.equals(s2)); // true
System.out.println(s1.equals(s3)); // false
```

Comparing Strings

3) Use the **`compareTo()`** method

```
s1.compareTo(s2)
```

- If `s1` is equal to `s2` → return **0**
- If `s1` is lexicographically less than `s2` → **negative integer offset**
  - The offset of the first two distinct characters in `s1` and `s2` from left to right
- If `s1` is lexicographically greater than `s2` → **positive integer offset**

```
String s1 = "abc";
String s2 = "abg";
System.out.println(s1.compareTo(s2));
```

-4

s1 is lexicographically less than s2
'c' is **4** less than 'g'

NOTE

- Syntax errors will occur if you compare strings by using relational operators such as >, >=, <, or <=.

```
c:\book>java OrderTwoCities
Enter the first city: Boston
Enter the second city: Atlanta
The cities in alphabetical order are Atlanta Boston
```

```java
import java.util.Scanner;

public class OrderTwoCities {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    // Prompt the user to enter two cities
    System.out.print("Enter the first city: ");
    String city1 = input.nextLine();
    System.out.print("Enter the second city: ");
    String city2 = input.nextLine();

    if (city1.compareTo(city2) < 0)
      System.out.println("The cities in alphabetical order are "
       + city1 + " " + city2);
    else
      System.out.println("The cities in alphabetical order are "
       + city2 + " " + city1);
  }
}
```

OrderTwoCities

Obtaining Substrings

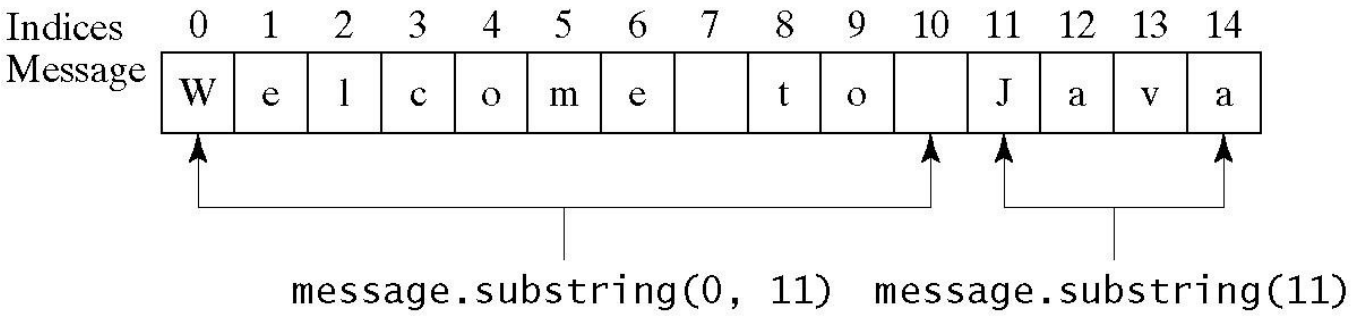
- Use the **`substring()`** method in the `String` class

TABLE 4.9 The `String` class contains the methods for obtaining substrings.

Method	*Description*
`substring(beginIndex)`	Returns this string's substring that begins with the character at the specified `beginIndex` and extends to the end of the string, as shown in Figure 4.2.
`substring(beginIndex, endIndex)`	Returns this string's substring that begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`, as shown in Figure 4.2. Note that the character at `endIndex` is not part of the substring.

```
String message = "Welcome to Java";
String message = message.substring(0, 11) + "HTML";
```

Welcome to HTML



Finding a Character or a Substring in a String

- The `String` class provides several versions of **`indexOf()`** and **`lastIndexOf()`** methods

TABLE 4.10 The `String` class contains the methods for finding substrings.

Method	*Description*
`index(ch)`	Returns the index of the first occurrence of `ch` in the string. Returns `-1` if not matched.
`indexOf(ch, fromIndex)`	Returns the index of the first occurrence of `ch` after `fromIndex` in the string. Returns `-1` if not matched.
`indexOf(s)`	Returns the index of the first occurrence of string `s` in this string. Returns `-1` if not matched.
`indexOf(s, fromIndex)`	Returns the index of the first occurrence of string `s` in this string after `fromIndex`. Returns `-1` if not matched.
`lastIndexOf(ch)`	Returns the index of the last occurrence of `ch` in the string. Returns `-1` if not matched.
`lastIndexOf(ch, fromIndex)`	Returns the index of the last occurrence of `ch` before `fromIndex` in this string. Returns `-1` if not matched.
`lastIndexOf(s)`	Returns the index of the last occurrence of string `s`. Returns `-1` if not matched.
`lastIndexOf(s, fromIndex)`	Returns the index of the last occurrence of string `s` before `fromIndex`. Returns `-1` if not matched.

indexOf()

- Returns the index of the **first occurrence** of character `ch` or string `s` in this string.
  - ✓ Return `-1` if not matched

```
"Welcome to Java".indexOf('W') returns
"Welcome to Java".indexOf('o') returns
"Welcome to Java".indexOf('o', 5) returns
"Welcome to Java".indexOf("come") returns
"Welcome to Java".indexOf("Java", 5) returns    .
"Welcome to Java".indexOf("java", 5) returns    .
```

lastIndexOf()

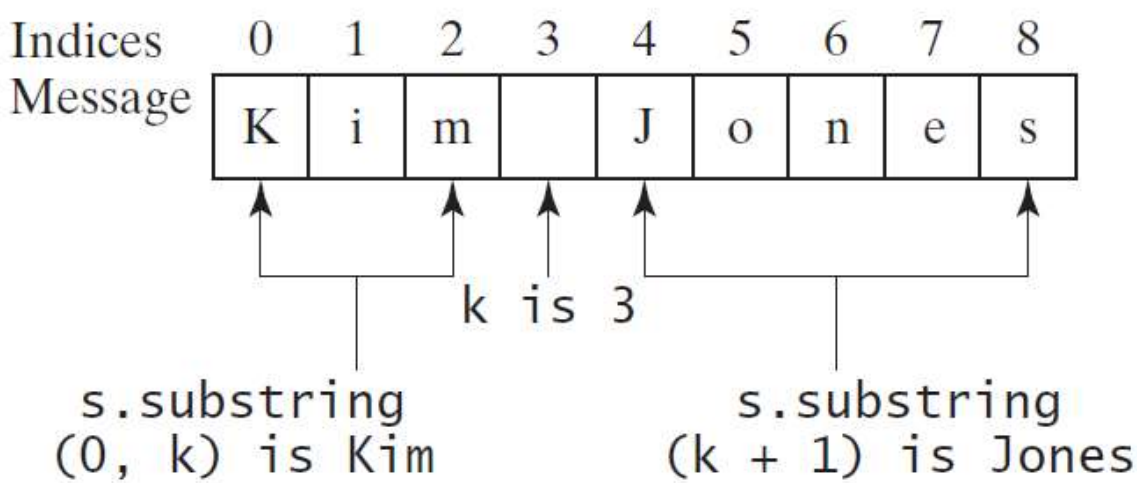
- Returns the index of the **last occurrence** of character `ch` or string `s` in this string.
  - ✓ Return `-1` if not matched

```
"Welcome to Java".lastIndexOf('W') returns
"Welcome to Java".lastIndexOf('o') returns
"Welcome to Java".lastIndexOf('o', 5) returns
"Welcome to Java".lastIndexOf("come") returns
"Welcome to Java".lastIndexOf("Java", 5) returns
"Welcome to Java".lastIndexOf("Java") returns
```

Finding a Character or a Substring in a String

- Example

```
String s = "Kim Jones"
int k = s.indexOf(' ');
String firstName = s.substring(0, k);
String lastName = s.substring(k + 1);
```



Conversion between Strings and Numbers

- To convert a `string` into an `int` value
  - ✓ use the **`Integer.parseInt()`** method

```
int intValue = Integer.parseInt(intString);
```

- To convert a `string` into a `double` value
  - ✓ use the **`Double.parseDouble()`** method

```
double doubleValue = Double.parseDouble(doubleString);
```

- Convert a number into a `string`
  - ✓ use the string concatenating operator

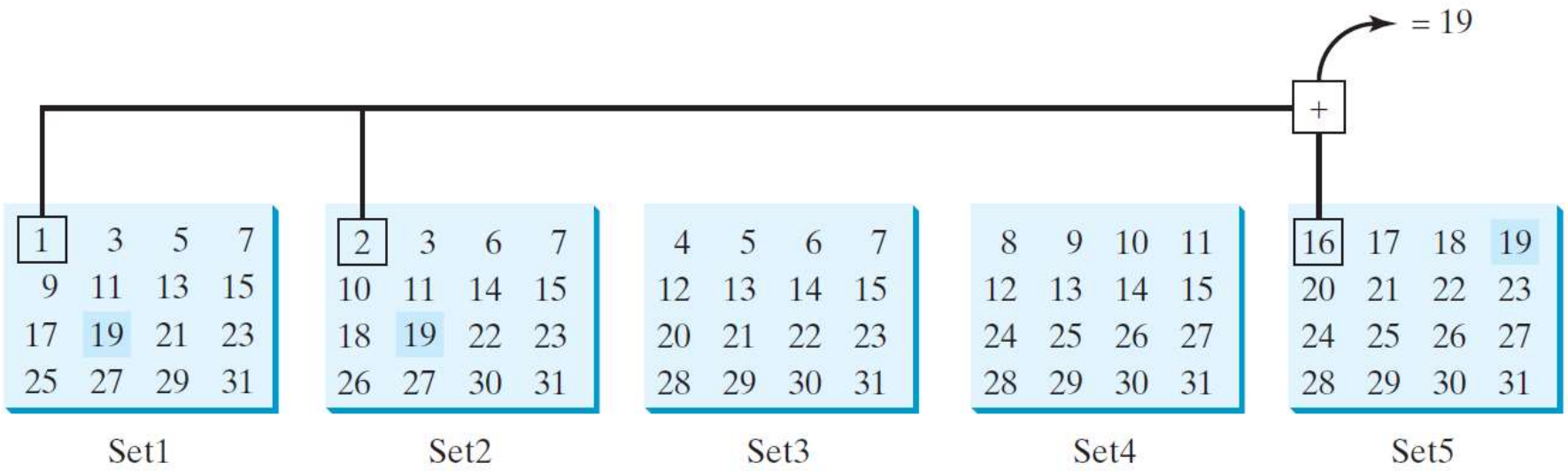
```
String s = number + "";
```

Check point

- Write answers to the problems below and explanations.
  - ✓ 4.16
  - ✓ 4.17
  - ✓ 4.20
  - ✓ 4.21
  - ✓ 4.23
  - ✓ 4.24

Problem: Guessing Birthday

- You can find out the date of the month when your friend was born by asking five questions.
  - ✓ Each question asks whether the day is in one of the five sets of numbers.



GuessBirthday

Mathematics Basis for the Game

- 19 is `10011` in binary. 7 is `111` in binary. 23 is `11101` in binary

```
  10000        00100        10000
     10           10         1000
+     1      +     1         100
-------      -------    +     1
  10011        00111       -------
                           11101
    ||           ||          ||
    19           7           23
```

$2^4 x1 + 2^3 x0 + 2^2 x0 + 2^1 x1 + 2^0 x1$
$= 16 + 0 + 0 + 2 + 1$
$= 19$

Decimal	Binary
1	00001
2	00010
3	00011
...	
19	10011
...	
31	11111

$$\begin{array}{r} b_5\ 0\ 0\ 0\ 0 \\ b_4\ 0\ 0\ 0 \\ b_3\ 0\ 0 \\ b_2\ 0 \\ +\quad b_1 \\ \hline b_5\, b_4\, b_3\, b_2\, b_1 \end{array} \qquad \begin{array}{r} 10000 \\ 10 \\ +\quad 1 \\ \hline 10011 \end{array} \qquad \begin{array}{r} 10000 \\ 1000 \\ 100 \\ 10 \\ +\quad 1 \\ \hline 11111 \end{array}$$

19 31

```
Is your birthday in Set1?
 1  3  5  7
 9 11 13 15
17 19 21 23
25 27 29 31
Enter 0 for No and 1 for Yes: 1 ↵Enter

Is your birthday in Set2?
 2  3  6  7
10 11 14 15
18 19 22 23
26 27 30 31
Enter 0 for No and 1 for Yes: 1 ↵Enter

Is your birthday in Set3?
 4  5  6  7
12 13 14 15
20 21 22 23
28 29 30 31
Enter 0 for No and 1 for Yes: 0 ↵Enter

Is your birthday in Set4?
 8  9 10 11
12 13 14 15
24 25 26 27
28 29 30 31
Enter 0 for No and 1 for Yes: 0 ↵Enter

Is your birthday in Set5?
16 17 18 19
20 21 22 23
24 25 26 27
28 29 30 31
Enter 0 for No and 1 for Yes: 1 ↵Enter
Your birthday is 19!
```

```java
import java.util.Scanner;
public class GuessBirthday {
 public static void main(String[] args)
  { String set1 =
      " 1  3  5  7\n" +
      " 9 11 13 15\n" +
      "17 19 21 23\n" +
      "25 27 29 31";
    String set2 =
      " 2  3  6  7\n" +
      "10 11 14 15\n" +
      "18 19 22 23\n" +
      "26 27 30 31";
    String set3 =
      " 4  5  6  7\n" +
      "12 13 14 15\n" +
      "20 21 22 23\n" +
      "28 29 30 31";
    String set4 =
      " 8  9 10 11\n" +
      "12 13 14 15\n" +
      "24 25 26 27\n" +
      "28 29 30 31";
    String set5 =
      "16 17 18 19\n" +
      "20 21 22 23\n" +
      "24 25 26 27\n" +
      "28 29 30 31";
int day = 0;
Scanner input=new Scanner(System.in);
System.out.print("Is your birthday inSet1?\n");
System.out.print(set1);
System.out.print("\nEnter 0 if No, 1 if Yes:");
int answer = input.nextInt();
if (answer == 1)   day += 1;
System.out.print("Is your birthday inSet2?\n");
System.out.print(set2);
System.out.print("\nEnter 0 if No, 1 if Yes:");
int answer = input.nextInt();
if (answer == 1)   day += 2;
System.out.print("Is your birthday inSet3?\n");
System.out.print(set3);
System.out.print("\nEnter 0 if No, 1 if Yes:");
int answer = input.nextInt();
if (answer == 1)   day += 4;
System.out.print("Is your birthday inSet4?\n");
System.out.print(set4);
System.out.print("\nEnter 0 if No, 1 if Yes:");
int answer = input.nextInt();
if (answer == 1)   day += 8;
System.out.print("Is your birthday inSet5?\n");
System.out.print(set5);
System.out.print("\nEnter 0 if No, 1 if Yes:");
int answer = input.nextInt();
if (answer == 1)   day += 16;
System.out.println("\nYour birthday is " + day +
"!");   }}
```

Converting a Hexadecimal Digit to a Decimal Value

- The hexadecimal number system has 16 digits: 0-9, A-F
  - ✓ A(10), B(11), C(12), D(13), E(14), F(15)

- Write a program that converts a hexadecimal digit into a decimal value.
  1) Read a hexadecimal digit (`ch`)
  2) Check if the input is exactly one character
  3) If yes, convert the hexadecimal digit into a decimal
     a) If `ch` is a letter between 'A' to 'F' → Convert the letter to decimal
        - `Decimal = ch – 'A' + 10`
     b) If `ch` is a digit → Dispay `ch`
     c) Otherwise, invalid input!

```java
import java.util.Scanner;
public class HexDigit2Dec {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter a hex digit: ");
    String hexString = input.nextLine();

    // Check if the hex string has exactly one
    if (hexString.length() != 1) {
      System.out.println("You must enter exactl
      System.exit(1);
    }

    // Display decimal value for the hex digit
    char ch = Character.toUpperCase(hexString.charAt(0));

    if (ch <= 'F' && ch >= 'A') {
      int value = ch - 'A' + 10;
      System.out.println("The decimal value for hex digit " + ch + " is "+ value);
    }
    else if (Character.isDigit(ch)) {
      System.out.println("The decimal value for hex digit " + ch + " is " + ch);
    }
    else {
      System.out.println(ch + " is an invalid input");
    }
  }
}
```

```
Enter a hex digit: AB7C ↵Enter
You must enter exactly one character
```

```
Enter a hex digit: B ↵Enter
The decimal value for hex digit B is 11
```

```
Enter a hex digit: 8 ↵Enter
The decimal value for hex digit 8 is 8
```

```
Enter a hex digit: T ↵Enter
T is an invalid input
```

HexDigit2Dec

Revising the Lottery Program Using `Strings`

- Write a program that randomly generates a lottery of a two-digit number, prompts the user to enter a two-digit number, and determines whether the user wins according to the following rule:
  - ✓ If the user input matches the lottery in exact order
    - the award is $10,000.
  - ✓ If the user input matches the lottery
    - the award is $3,000.
  - ✓ If one digit in the user input matches a digit in the lottery
    - the award is $1,000.
- Rewrites the lottery program in Listing 3.7 using strings. Using strings simplifies this program.

Lottery program

```
import java.util.Scanner;
public class Lottery {
  public static void main(String[] args) {
    int lottery = (int)(Math.random() * 100);

    Scanner input = new Scanner(System.in);
    System.out.print("Enter your lottery pick (two digits): ");
    int guess = input.nextInt();

    int lotteryDigit1 = lottery / 10;
    int lotteryDigit2 = lottery % 10;

    int guessDigit1 = guess / 10;
    int guessDigit2 = guess % 10;

    System.out.println("The lottery number is " + lottery);

    if (guess == lottery)
        System.out.println("Exact match: you win $10,000");
    else if (guessDigit2 == lotteryDigit1 && guessDigit1 == lotteryDigit2)
        System.out.println("Match all digits: you win $3,000");
    else if (guessDigit1==lotteryDigit1 || guessDigit1==lotteryDigit2 || guessDigit2==lotteryDigit1 || guessDigit2==lotteryDigit2)
        System.out.println("Match one digit: you win $1,000");
    else
        System.out.println("Sorry, no match");
  }
}
```

```java
import java.util.Scanner;
public class LotteryUsingStrings {
  public static void main(String[] args) {
    String lottery = "" + (int)(Math.random() * 10) + (int)(Math.random() * 10);

    Scanner input = new Scanner(System.in);
    System.out.print("Enter your lottery pick (two digits): ");
    String guess = input.nextLine();
    char lotteryDigit1 = lottery.charAt(0);
    char lotteryDigit2 = lottery.charAt(1);
    char guessDigit1 = guess.charAt(0);
    char guessDigit2 = guess.charAt(1);

    System.out.println("The lottery number is " + lottery);

    if (guess.equals(lottery))
          System.out.println("Exact match: you win $10,000");
    else if (guessDigit2 == lotteryDigit1 && guessDigit1 == lotteryDigit2)
          System.out.println("Match all digits: you win $3,000");
    else if (guessDigit1 == lotteryDigit1 || guessDigit1 == lotteryDigit2 ||
             guessDigit2 == lotteryDigit1 || guessDigit2 == lotteryDigit2)
          System.out.println("Match one digit: you win $1,000");
    else
          System.out.println("Sorry, no match");
  }
}
```

```
Enter your lottery pick: 23 ↵Enter
The lottery number is 14
Sorry: no match
```

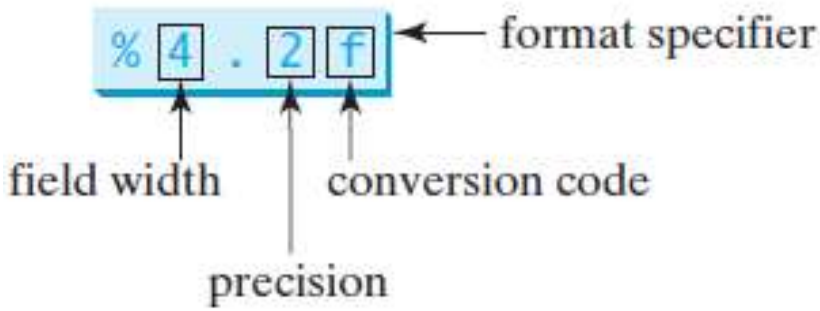
LotteryUsingStrings

Formatting Output

- Use **`System.out.printf()`** method to display formatted output on the console

```
System.out.printf(format, item1, item2, …, item);
```

  - ✓ `format`: A string that may consist of substrings and *format specifiers*.



    - A ***format specifier*** specifies how an item should be displayed.
      - An `item` may be a **numeric** value, **character**, **boolean** value, or a **string**.
      - Each *specifier* begins with a percent sign (**%**).

Frequently-Used Specifiers

TABLE 4.11 Frequently Used Format Specifiers

Format Specifier	*Output*	*Example*
%b	a Boolean value	true or false
%c	a character	'a'
%d	a decimal integer	200
%f	a floating-point number	45.460000
%e	a number in standard scientific notation	4.556000e+01
%s	a string	"Java is cool"

```
int count = 5;
double amount = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);
```

items

```
display          count is 5 and amount is 45.560000
```

Frequently-Used Specifiers

TABLE 4.12 Examples of Specifying Width and Precision

Example	*Output*
`%5c`	Output the character and add four spaces before the character item, because the width is 5.
`%6b`	Output the Boolean value and add one space before the false value and two spaces before the true value.
`%5d`	Output the integer item with width at least 5. If the number of digits in the item is < 5, add spaces before the number. If the number of digits in the item is > 5, the width is automatically increased.
`%10.2f`	Output the floating-point item with width at least 10 including a decimal point and two digits after the point. Thus, there are 7 digits allocated before the decimal point. If the number of digits before the decimal point in the item is < 7, add spaces before the number. If the number of digits before the decimal point in the item is > 7, the width is automatically increased.
`%10.2e`	Output the floating-point item with width at least 10 including a decimal point, two digits after the point and the exponent part. If the displayed number in scientific notation has width less than 10, add spaces before the number.
`%12s`	Output the string with width at least 12 characters. If the string item has fewer than 12 characters, add spaces before the string. If the string item has more than 12 characters, the width is automatically increased.

Formatting Output

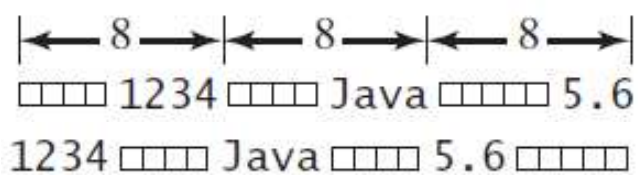
- Examples

1234 #Java #51.67

```
System.out.printf("%3d#%2s#%4.2f\n", 1234, "Java", 51.6653);
```

- By default, the output is ***right justified***.
  - ✓ You should put the minus sign (**-**) in the *format specifier* to specify that the item is *left justified* in the output.

```
System.out.printf("%8d%8s%8.1f\n", 1234, "Java", 5.63);
System.out.printf("%-8d%-8s%-8.1f \n", 1234, "Java", 5.63);
```



```java
public class FormatDemo {
  public static void main(String[] args) {
    // Display the header of the table
    System.out.printf("%-10s%-10s%-10s%-10s%-10s\n", "Degrees",
      "Radians", "Sine", "Cosine", "Tangent");

    // Display values for 30 degrees
    int degrees = 30;
    double radians = Math.toRadians(degrees);
    System.out.printf("%-10d%-10.4f%-10.4f%-10.4f%-10.4f\n", degrees,
      radians, Math.sin(radians), Math.cos(radians), Math.tan(radians));

    // Display values for 60 degrees
    degrees = 60;
    radians = Math.toRadians(degrees);
    System.out.printf("%-10d%-10.4f%-10.4f%-10.4f%-10.4f\n", degrees,
      radians, Math.sin(radians), Math.cos(radians), Math.tan(radians));
  }
}
```

Degrees	Radians	Sine	Cosine	Tangent
30	0.5236	0.5000	0.8660	0.5773
60	1.0472	0.8660	0.5000	1.7320

FormatDemo

Exception Handling-Motivation

- When a program runs into a runtime error, the program terminates abnormally.

How can you handle the runtime error
so that the program can continue to run or terminate
gracefully?

Exception-Handling Overview

- Show runtime error

  Quotient

- Fix it using an `if` statement

  QuotientWithIf

- With a method

  QuotientWithMethod

Exception Advantages

- **Exception Handling** enables a **method** to throw an **exception** to its **caller**.

- Handling **`ArithmeticException`**

QuotientWithException

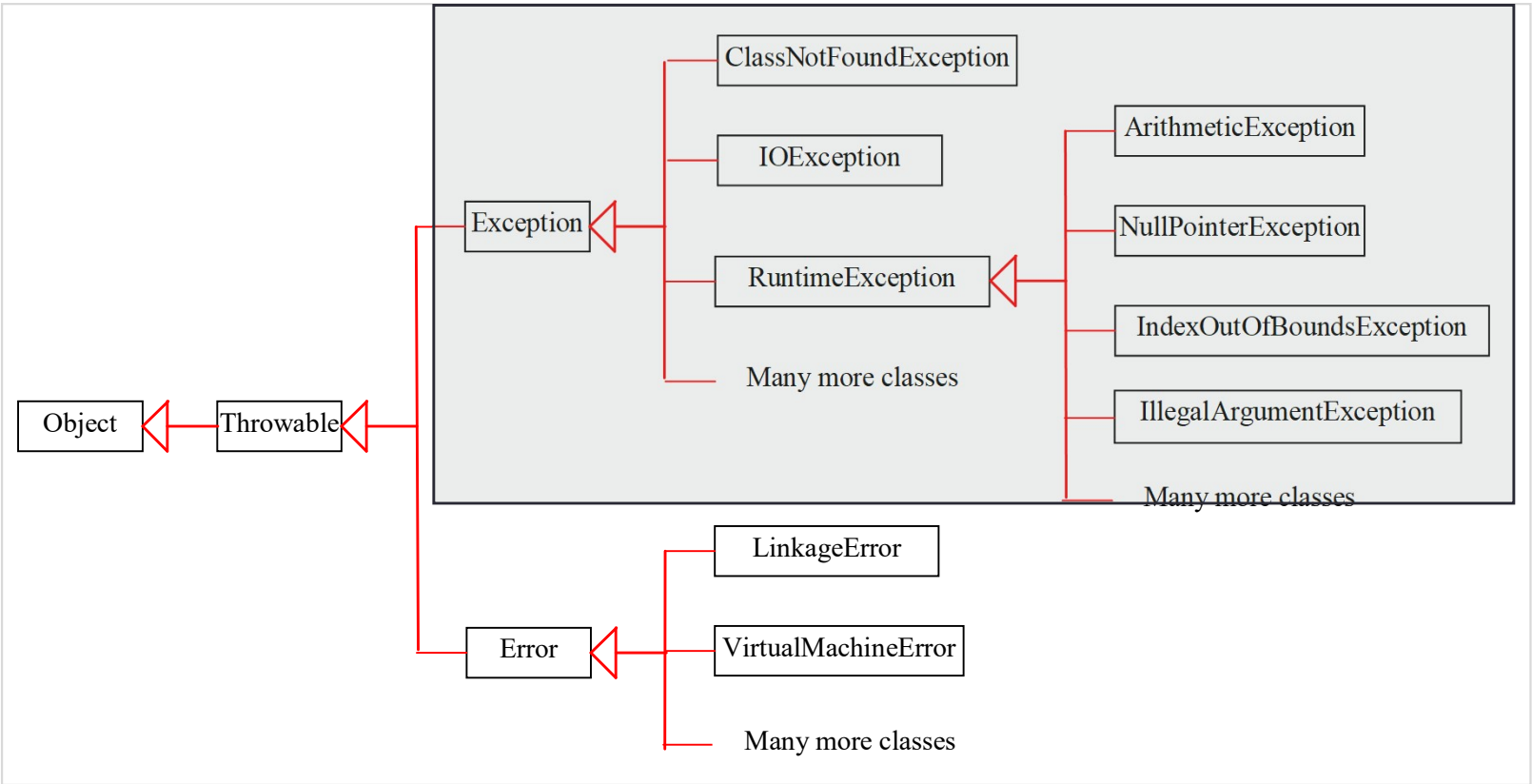
Handling `InputMismatchException`

- By handling `InputMismatchException`, your program will continuously read an input until it is correct.

InputMismatchExceptionDemo

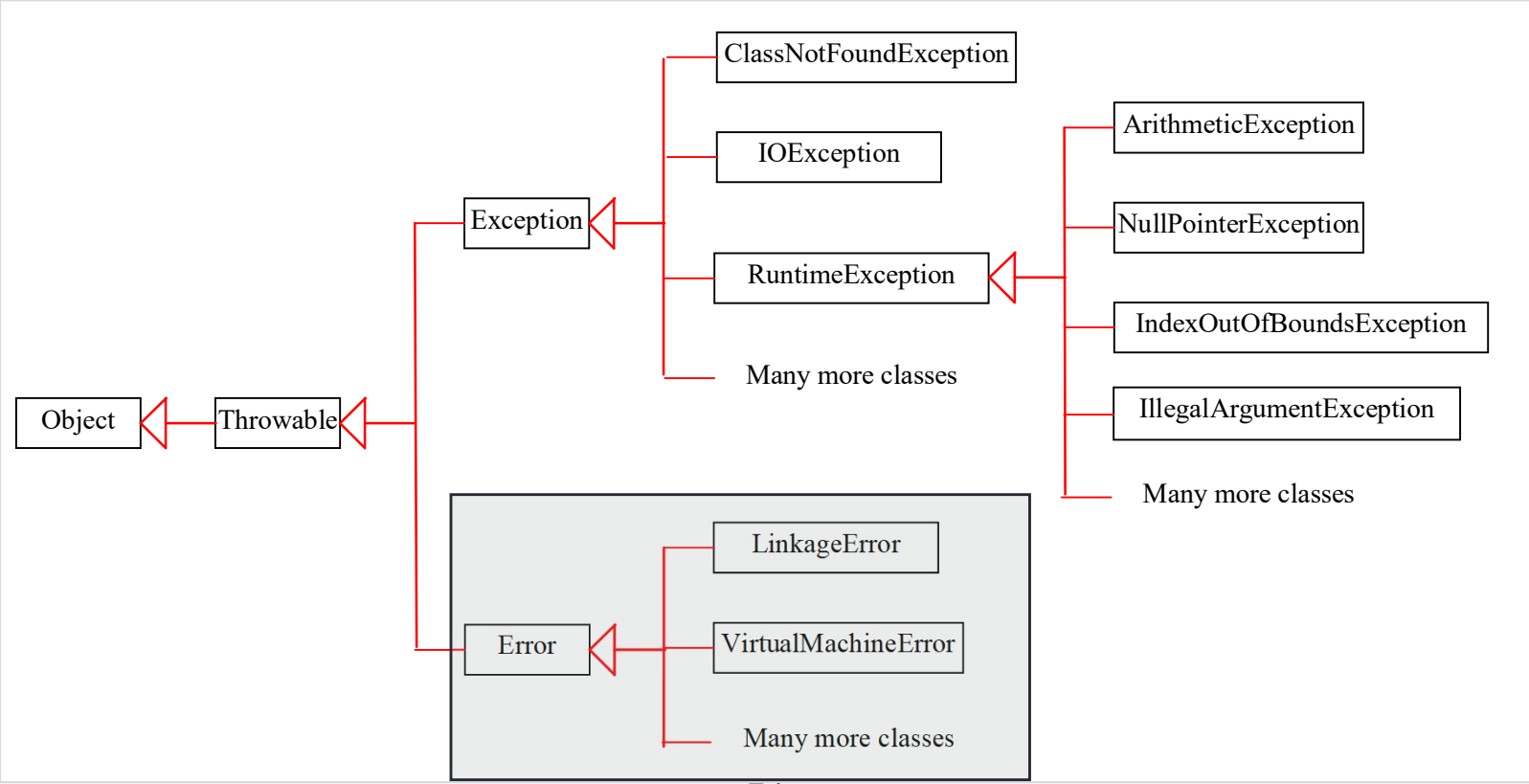
Exception

- describes errors caused by your program and external circumstances
  - ✓ caught and handled by your program



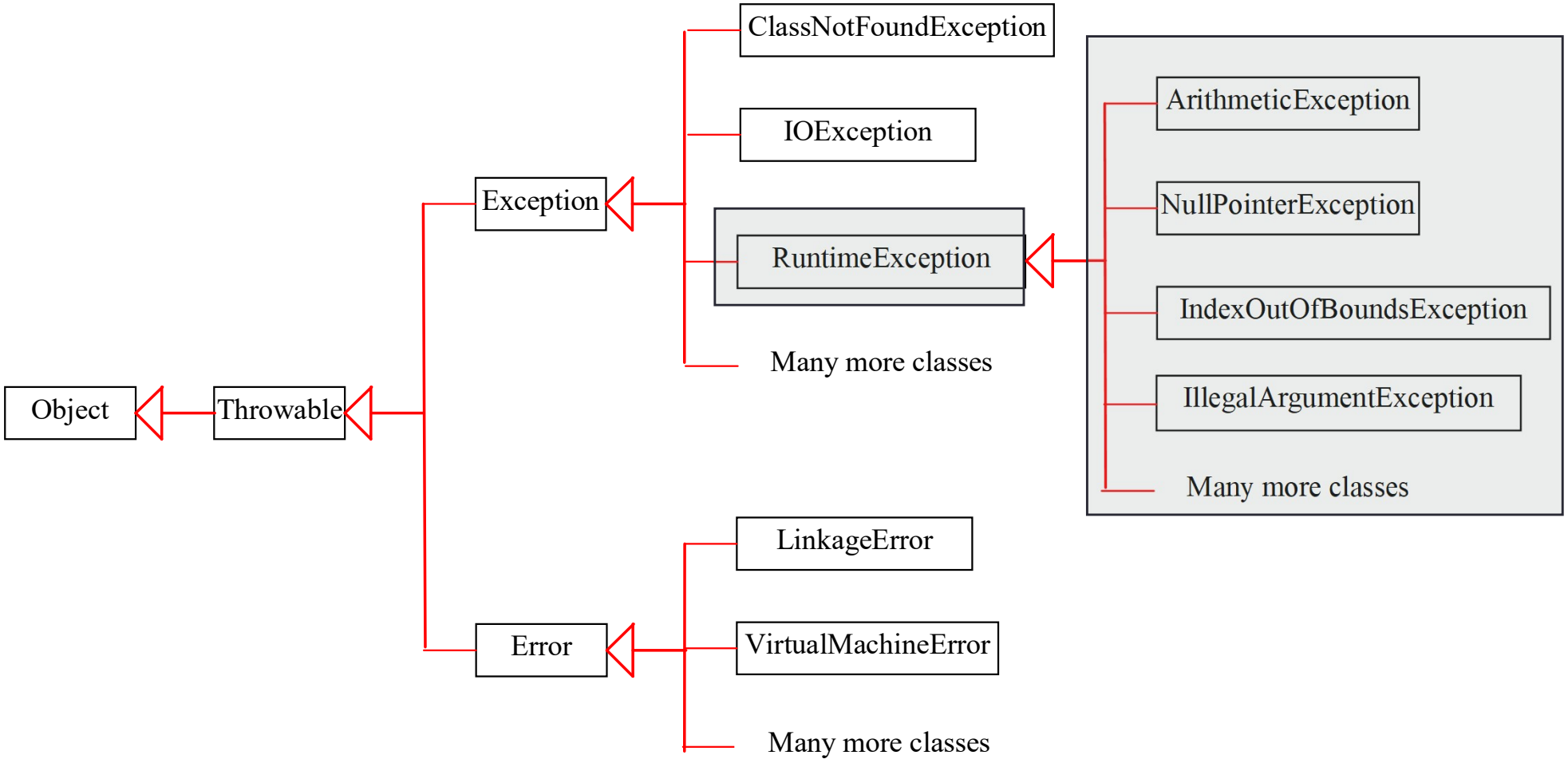
System **Errors**

- thrown by **JVM**
- represented in the `Error` class describing **internal system errors**.



Runtime **Exceptions**

- caused by **programming errors**

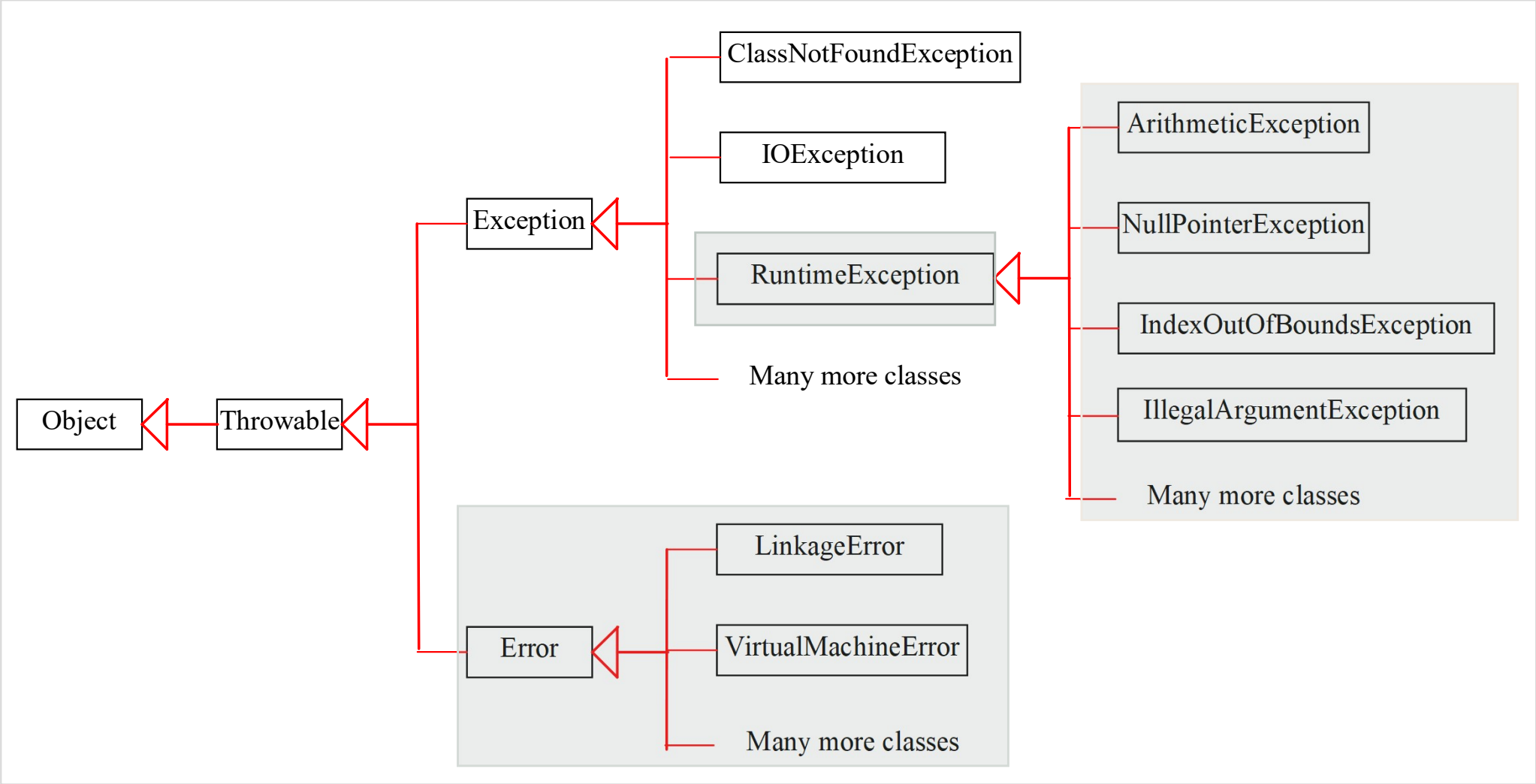


Checked VS **Unchecked** Exceptions

- **Unchecked** exceptions
  - ✓ Ex) **Runtime Exception**, **Error** and their subclasses

- **Checked** exceptions
  - ✓ The **compiler** forces the **programmer** to **check** and deal with the exceptions.
    - ➢ Ex) all other exceptions

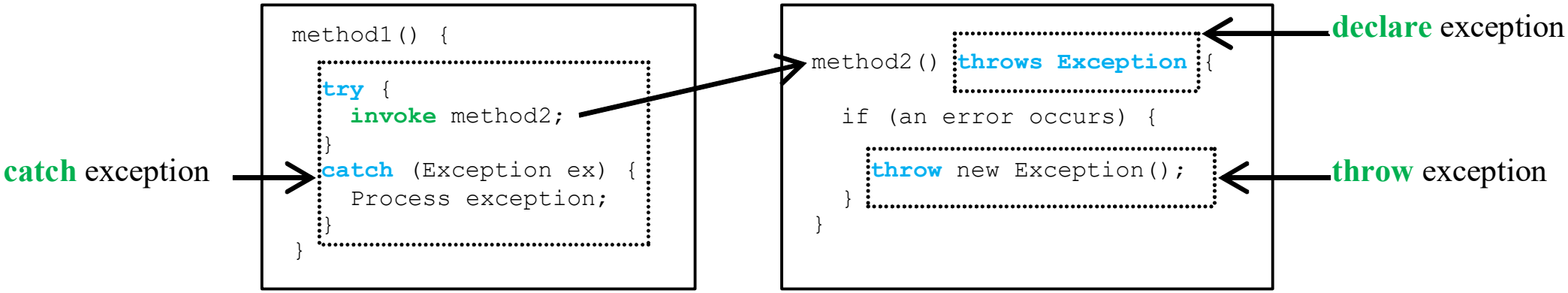
Unchecked Exceptions



Unchecked Exceptions

- **Java** does **not mandate** to write code to **catch** unchecked exceptions.
- reflect programming **logic errors**
  - ✓ **`NullPointerException`**
    - If you **access** an **object** through a **reference** variable **before** an object is **assigned** to it;
  - ✓ **`IndexOutOfBoundsException`**
    - If you **access** an **element** in an array **outside** the **bounds** of the array.

Declaring/Throwing/Catching Exceptions



1) Declaring Exceptions

- Every method must state the **types** of checked **exceptions** it **might throw**

```
public void myMethod() throws IOException
```

```
public void myMethod() throws IOException, OtherException
```

2) Throwing Exceptions

- When the program a) **detect** an error, the program can b) **create** an **instance** of an appropriate **exception** type and c) **throw** it.

```
throw new TheException();
```

||

```
TheException ex = new TheException();
throw ex;
```

Throwing Exceptions Example

```
/** Set a new radius */
 public void setRadius(double newRadius) throws
                                IllegalArgumentException {
   if (newRadius >= 0)
     radius =  newRadius;
   else
     throw new IllegalArgumentException(
                    "Radius cannot be negative");
 }
```

3) Catching Exceptions

```
try {
  statements; //Statements that may throw exceptions
}
catch (Exception1 exVar1) {
  handler for exception1;
}
catch (Exception2 exVar2) {
  handler for exception2;
}
...
catch (ExceptionN exVarN) {
  handler for exceptionN;
}
```

Practice

- Write and run two example programs, `QuotientWithException` (Slide #6) and `InputMismatchExceptionDemo` (Slide #7) programs

Catching Exceptions

```
main method {
  ...
  try {
    ...
    invoke method1;
    statement1;
  }
  catch (Exception1 ex1) {
    Process ex1;
  }
  statement2;
}
```

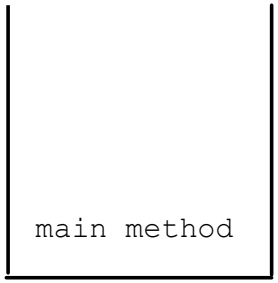
```
method1 {
  ...
  try {
    ...
    invoke method2;
    statement3;
  }
  catch (Exception2 ex2) {
    Process ex2;
  }
  statement4;
}
```

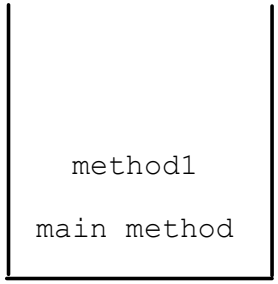
```
method2 {
  ...
  try {
    ...
    invoke method3;
    statement5;
  }
  catch (Exception3 ex3) {
    Process ex3;
  }
  statement6;
}
```

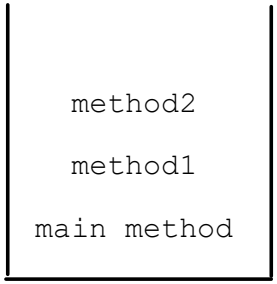
An exception is thrown in method3

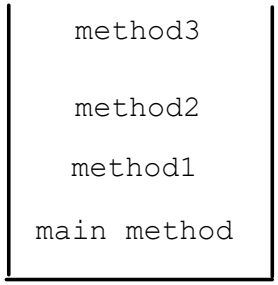
Exception3

Call Stack









Catch|Declare Checked Exceptions

- Suppose **`p2()`** is defined as follows:

```
void p2() throws IOException {
  if (a file does not exist) {
     throw new IOException("File does not exist");
  }

  ...
}
```

Catch|Declare Checked Exceptions

- If a method declares a **checked** exception, you must
  1) declare to throw the exception in the calling
  2) invoke it in a **`try-catch`** block, or
     - Suppose that **`p1()`** invokes method **`p2()`** that may throw a checked exception (**`IOException`**)

```
void p1() {
  try {
    p2();
  }
  catch (IOException ex) {
    ...
  }
}
```

(a)

```
void p1() throws IOException {

  p2();

}
```

(b)

Examples

- Modifying the **`setRadius()`** method in the **`Circle`** class
  - ✓ The new **`setRadius()`** method throws an exception if radius is negative.

CircleWithException

TestCircleWithException

Rethrowing Exceptions

```
try {
  statements;
}
catch(TheException ex) {
  perform operations before exits;
  throw ex;
}
```

The `finally` Clause

```
try {
   statements;
}
catch(TheException ex)  {
   handling ex;
}
finally {
   finalStatements;
}
```

Trace a Program Execution

```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

Suppose no exceptions in the statements

animation

Trace a Program Execution

```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

The final block is **always** executed

animation

Trace a Program Execution

```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

Next statement in the method is executed

animation

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

Suppose an exception of type **Exception1** is thrown in statement1

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

The exception is handled.

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

The **final** block is **always** executed.

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

The **next** statement in the method is now executed.

animation

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

statement2 throws an exception of type Exception2.

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

Handling exception

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  }
finally {
  finalStatements;
}

Next statement;
```

Execute the final block

Cautions

- Exception handling separates **error-handling** code from **normal programming** tasks
  - ✓ Making programs easier to read and to modify
  - ✓ requires **more time** and **resources**
    - ➢ Ex) instantiating a new exception object, rolling back the call stack, and propagating the errors to the calling methods.

When to Throw Exceptions

- If you want the **exception** to be **processed** by its **caller**, you should create an exception object and throw it.

- If you can handle the exception in the method where it occurs, there is **no need** to throw it.

Practice

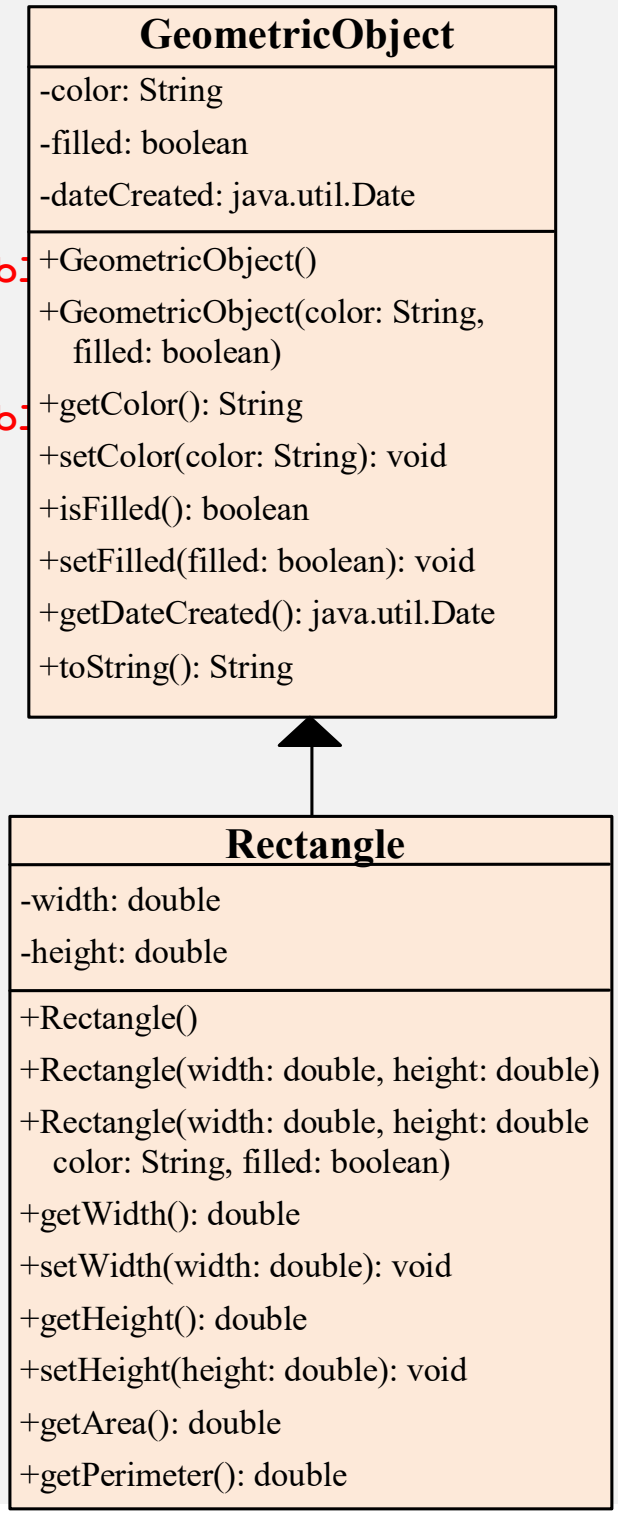
- Write `RectangleWithException` class by modify two constructors, `RectangleFromSimpleGeometricObject(double width, double height)` and `RectangleFromSimpleGeometricObject (double width, double height, String color, boolean filled)`, to throw an **exception** when having any negative value of arguments.
- Write a test class that tries to create a rectangle with a negative value of width or height

- Submit 1) **java files** and
  2) **screenshot of test result**

Rectangle
-width: double -height: double
+Rectangle() +Rectangle(width: double, height: double) +Rectangle(width: double, height: double color: String, filled: boolean) +getWidth(): double +setWidth(width: double): void +getHeight(): double +setHeight(height: double): void +getArea(): double +getPerimeter(): double

```java
public class SimpleGeometricObject {
  private String color = "white";
  private boolean filled;
  private java.util.Date dateCreated;

public SimpleGeometricObject() {
    dateCreated = new java.util.Date();   }
public SimpleGeometricObject(String color, boolean filled) {
    dateCreated = new java.util.Date();
    this.color = color;
    this.filled = filled;  }

  public String getColor() {
    return color;  }

  public void setColor(String color) {
    this.color = color;  }

  public boolean isFilled() {
    return filled;  }

  public void setFilled(boolean filled) {
    this.filled = filled;  }

  public java.util.Date getDateCreated() {
    return dateCreated;  }

  public String toString() {
    return "created on " + dateCreated + "\ncolor: " + color +
      " and filled: " + filled;  }   }
```

```java
public class RectangleFromSimpleGeometricObject extends SimpleGeometricObject {
  private double width;
  private double height;

public RectangleFromSimpleGeometricObject() {
}
public RectangleFromSimpleGeometricObject(double width, doubl
    this.width = width;
    this.height = height; }
public RectangleFromSimpleGeometricObject(double width, doubl
        String color, boolean filled) {
    this.width = width;
    this.height = height;
    setColor(color);
    setFilled(filled);  }

  public double getWidth() {
    return width;  }
  public void setWidth(double width) {
    this.width = width; }

  public double getHeight() {
    return height;  }
  public void setHeight(double height) {
    this.height = height;  }

  public double getArea() {
    return width * height;  }

  public double getPerimeter() {
    return 2 * (width + height);  }}
```



The `File` Class

- provide an **abstraction** that deals with most of the machine-dependent complexities of **files** and **path names** in a machine-independent fashion.
  - ✓ The **filename** is a **string**.

java.io.File	
`+File(pathname: String)`	Creates a `File` object for the specified path name. The path name may be a directory or a file.
`+File(parent: String, child: String)`	Creates a `File` object for the child under the directory parent. The child may be a file name or a subdirectory.
`+File(parent: File, child: String)`	Creates a `File` object for the child under the directory parent. The parent is a `File` object. In the preceding constructor, the parent is a string.
`+exists(): boolean`	Returns true if the file or the directory represented by the `File` object exists.
`+canRead(): boolean`	Returns true if the file represented by the `File` object exists and can be read.
`+canWrite(): boolean`	Returns true if the file represented by the `File` object exists and can be written.
`+isDirectory(): boolean`	Returns true if the `File` object represents a directory.
`+isFile(): boolean`	Returns true if the `File` object represents a file.
`+isAbsolute(): boolean`	Returns true if the `File` object is created using an absolute path name.
`+isHidden(): boolean`	Returns true if the file represented in the `File` object is hidden. The exact definition of *hidden* is system-dependent. On Windows, you can mark a file hidden in the File Properties dialog box. On Unix systems, a file is hidden if its name begins with a period(.) character.
`+getAbsolutePath(): String`	Returns the complete absolute file or directory name represented by the `File` object.
`+getCanonicalPath(): String`	Returns the same as `getAbsolutePath()` except that it removes redundant names, such as "." and "..", from the path name, resolves symbolic links (on Unix), and converts drive letters to standard uppercase (on Windows).
`+getName(): String`	Returns the last name of the complete directory and file name represented by the `File` object. For example, new `File("c:\\book\\test.dat").getName()` returns `test.dat`.
`+getPath(): String`	Returns the complete directory and file name represented by the `File` object. For example, new `File("c:\\book\\test.dat").getPath()` returns `c:\book\test.dat`.
`+getParent(): String`	Returns the complete parent directory of the current directory or the file represented by the `File` object. For example, new `File("c:\\book\\test.dat").getParent()` returns `c:\book`.
`+lastModified(): long`	Returns the time that the file was last modified.
`+length(): long`	Returns the size of the file, or 0 if it does not exist or if it is a directory.
`+listFile(): File[]`	Returns the files under the directory for a directory `File` object.
`+delete(): boolean`	Deletes the file or directory represented by this `File` object. The method returns true if the deletion succeeds.
`+renameTo(dest: File): boolean`	Renames the file or directory represented by this `File` object to the specified name represented in dest. The method returns true if the operation succeeds.
`+mkdir(): boolean`	Creates a directory represented in this `File` object. Returns true if the the directory is created successfully.
`+mkdirs(): boolean`	Same as `mkdir()` except that it creates directory along with its parent directories if the parent directories do not exist.

Explore File Properties

- Write a program that **creates** files and **obtain** their **properties**.

Command Prompt

```
C:\book>java TestFileClass
Does it exist? true
Can it be read? true
Can it be written? true
Is it a directory? false
Is it a file? true
Is it absolute? false
Is it hidden? false
What is its absolute path? C:\book\.\image\us.gif
What is its canonical path? C:\book\image\us.gif
What is its name? us.gif
What is its path? .\image\us.gif
When was it last modified? Sat May 08 14:00:34 EDT 1999
What is the path separator? ;
What is the name separator? \

C:\book>
```

Command Prompt - telnet panda

```
$ pwd
/home/liang/book
$ java TestFileClass
Does it exist? true
Can it be read? true
Can it be written? true
Is it a directory? false
Is it a file? true
Is it absolute? false
Is it hidden? false
What is its absolute path? /home/liang/book/./image/us.gif
What is its canonical path? /home/liang/book/image/us.gif
What is its name? us.gif
What is its path? ./image/us.gif
When was it last modified? Wed Jan 23 11:00:14 EST 2002
What is the path separator? :
What is the name separator? /
$
```

TestFileClass

Text Input and Output

- create `File` objects using appropriate Java I/O classes like the `Scanner` and `PrintWriter` classes.
  - ✓ the `Scanner` class
    - to **read strings** and **numeric** values from a text file
  - ✓ the `PrintWriter` class
    - to **write strings** and **numeric** values to a text file

Writing Data Using `PrintWriter`

java.io.PrintWriter	WriteData
+PrintWriter(filename: String)	Creates a PrintWriter for the specified file.
+print(s: String): void	Writes a **string**.
+print(c: char): void	Writes a **character**.
+print(cArray: char[]): void	Writes an **array of character**.
+print(i: int): void	Writes an **int** value.
+print(l: long): void	Writes a **long** value.
+print(f: float): void	Writes a **float** value.
+print(d: double): void	Writes a **double** value.
+print(b: boolean): void	Writes a **boolean** value.
Also contains the overloaded **println**() methods.	A println method acts like a print method; additionally it prints a line separator. The line separator string is defined by the system. It is \r\n on Windows and \n on Unix.
Also contains the overloaded **printf**() methods.	The printf method was introduced in §4.6, "Formatting Console Output and Strings."

Reading Data Using `Scanner`

ReadData

java.util.Scanner	
+Scanner(source: File)	Creates a Scanner object to read data from the specified file.
+Scanner(source: String)	Creates a Scanner object to read data from the specified string.
+**close**()	Closes this scanner.
+hasNext(): boolean	Returns true if this scanner has another token in its input.
+next(): String	Returns next token as a **string**.
+nextByte(): byte	Returns next token as a **byte**.
+nextShort(): short	Returns next token as a **short**.
+nextInt(): int	Returns next token as an **int**.
+nextLong(): long	Returns next token as a **long**.
+nextFloat(): float	Returns next token as a **float**.
+nextDouble(): double	Returns next token as a **double**.
+useDelimiter(pattern: String): Scanner	Sets this scanner's delimiting pattern.

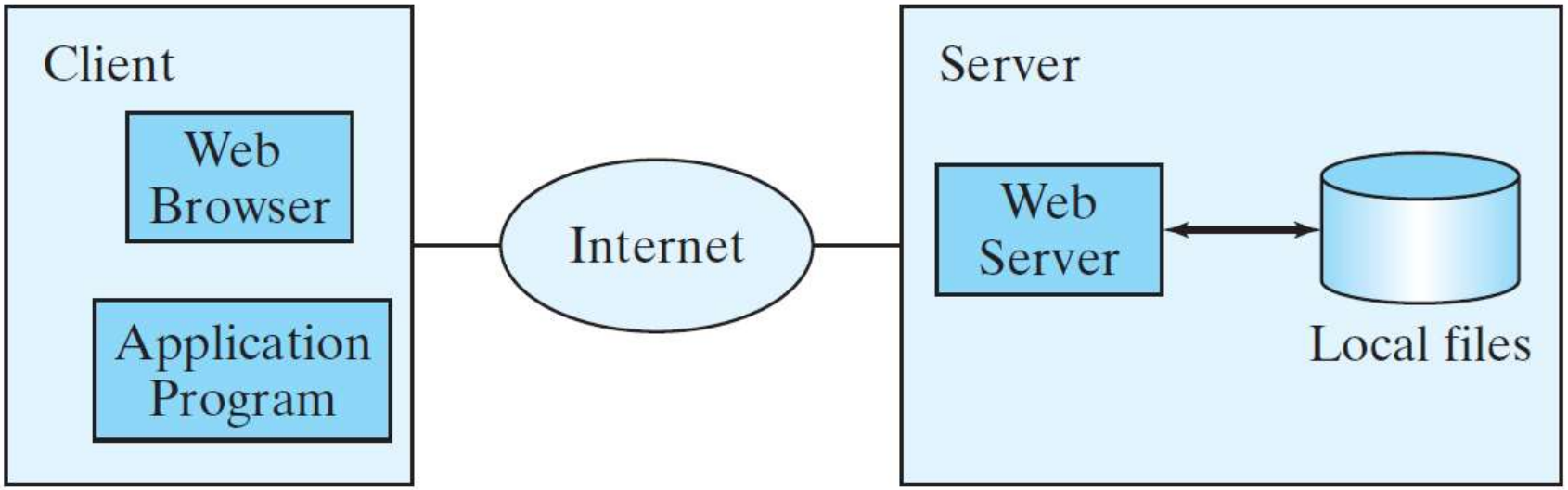
Practice

- Write a test class that
  1) creates a `file` object
  2) writes **names**, **class**, and **majors** of two CIS cadets (information for a cadet in a line) to the file
  3) and then read the file and display the information of two cadets (one cadet information in a line).

- Submit 1) **java files** and 2) **screenshot of test result**

Reading Data from the Web

- Just like you can read data from a file on your computer, you can **read data** from a **file** on the **Web**.



Reading Data from the Web

1. **Create** a `URL` object

```
URL url = new URL("www.google.com/index.html");
```

2. use the `openStream()` method defined in the `URL` class to open an input stream and use this stream to create a `Scanner` object as follows

```
Scanner input = new Scanner(url.openStream());
```