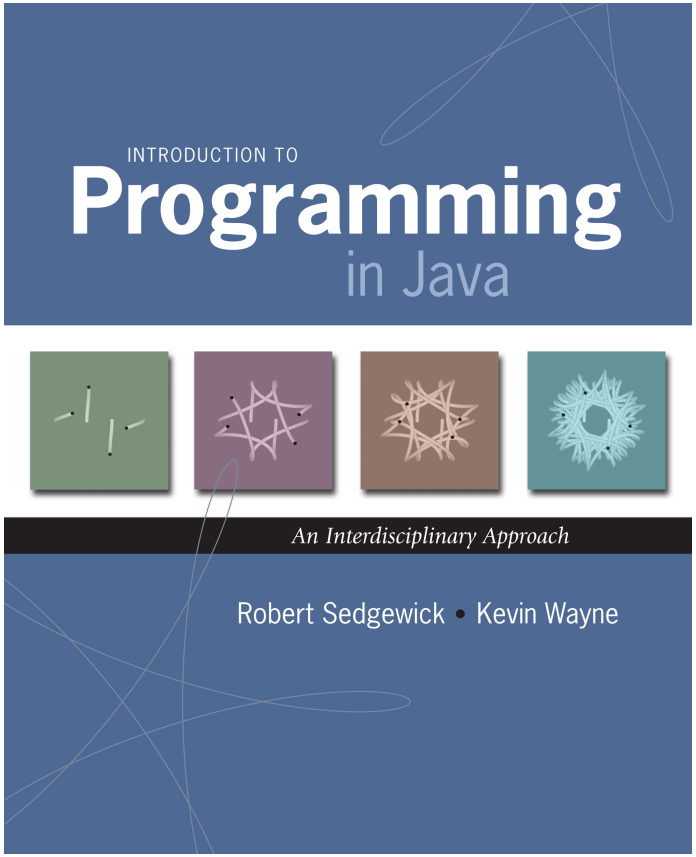
Transitioning to Java II: Strings, Files, and Functions



Strings

Strings: Example Program

/Users/bjbrown/introcs/Ruler.java

New Open Save Close Cut Copy Paste Undo

```
1  public class Ruler {
2     public static void main(String[] args) {
3        String ruler1 = "1";
4        String ruler2 = ruler1 + " 2 " + ruler1;
5        String ruler3 = ruler2 + " 3 " + ruler2;
6        String ruler4 = ruler3 + " 4 " + ruler3;
7        System.out.println(ruler4);
8     }
9  }
10
```

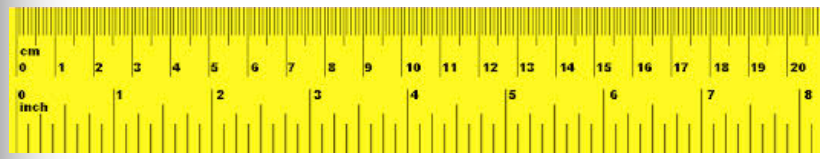
Interactions | Console | Compiler Output

```
Welcome to DrJava.  Working directory is /Users/bjbrown/introcs
> java Ruler
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
>
```

Editing /Users/bjbrown/introcs/Ruler.java 9:0

Download Ruler.java from booksite, section 1.2

1 2 1 3 1 2 1 4 1 2 1 3 1 2 1



Penn Engineering

Strings

- Strings are really arrays of chars

```
String name = ”M. Smith";
System.out.println(name.charAt(3));      // S
```

index	0	1	2	3	4	5	6	7
char	M	.		S	m	i	t	h

- Once a String is created it cannot be changed
  - Strings are *immutable*
  - Therefore, we must reassign it. For example:

```
String s = ”how i met your mother";
s = s.toUpperCase();
System.out.println(s);    // HOW I MET YOUR MOTHER
```

`String` methods

Method name	Description
`indexOf(`**`str`**`)`	index where the start of the given string appears in this string (-1 if it is not there)
`length()`	number of characters in this string

`substring(`**`index1`**`, `**`index2`**`)`
or
`substring(`**`index1`**`)`
	the characters in this string from *index1* (inclusive) to *index2* (exclusive);
	if *index2* omitted, grabs till end of string

`toLowerCase()`	a new string with all lowercase letters
`toUpperCase()`	a new string with all uppercase letters

- These methods are called using the dot notation:

```
String test = "Hello world";
System.out.println(test.length());   // 11
```

String Method Examples

```
//              01234567890123
String s1 = "Hello CIS 110!";
System.out.println(s1.length());          // 14
System.out.println(s1.indexOf("e"));      // 1
System.out.println(s1.substring(6, 9))    // "CIS"

String s2 = s1.substring(6, 13);
System.out.println(s2.toLowerCase());     // "cis 110"
```

- How would you extract the first word from ANY string?

Strings as parameters

```
public class StringParameters {
    public static void main(String[] args) {
        sayHello(”Lily");

        String lawyer = ”Marshall";
        sayHello(lawyer);
    }

    public static void sayHello(String name) {
        System.out.println("Welcome, " + name);
    }
}
```

append

Output:

```
Welcome, Lily
Welcome, Marshall
```

Comparing Strings

- Relational operators such as < and == fail on Strings.

```
public static void main(String[] args) {
  String name = args[0];
  if (name == "Barney") {
    System.out.println("It's gonna be legend- (wait for it)");
    System.out.println("-dary!");
  }
}
```

  - This code will compile, but it will not print the phrase.

  - == compares objects by *references*, so it often gives `false` even when two `String`s have the same letters.

Comparing Strings

- Objects are compared using a method named `equals`.

```
public static void main(String[] args) {
  String name = args[0];
  if (name.equals("Barney")) {
    System.out.println("It's gonna be legend- (wait for it)");
    System.out.println("-dary!");
  }
}
```

Method	Description
`equals(`**str**`)`	whether two strings contain the same characters
`equalsIgnoreCase(`**str**`)`	whether two strings contain the same characters, ignoring upper vs. lower case
`startsWith(`**str**`)`	whether one contains other's characters at start
`endsWith(`**str**`)`	whether one contains other's characters at end
`contains(`**str**`)`	whether the given string is found within this one

`String/char` question

- A *Caesar cipher* is a simple encryption where a message is encoded by shifting each letter by a given amount.
  - e.g. with a shift of 3, A → D, H → K, X → A, and Z → C

- Write a program that reads a message from the user and performs a Caesar cipher on its letters:

```
Your secret message: Brad thinks Angelina is cute
Your secret key: 3
The encoded message: eudg wklqnv dqjholqd lv fxwh
```

Caesar Cipher 1

```
// This program reads a message and a secret key from the user and
// encrypts the message using a Caesar cipher, shifting each letter.

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

        System.out.print("Your secret message: ");
        String message = StdIn.readLine();
        message = message.toLowerCase();

        System.out.print("Your secret key: ");
        int key = StdIn.readInt();

        encode(message, key);
    }

    ...
```

Caesar Cipher 2

```
// This method encodes the given text string using a Caesar
// cipher, shifting each letter by the given number of places.
public static void encode(String text, int shift) {
    System.out.print("The encoded message: ");
    for (int i = 0; i < text.length(); i++) {
        char letter = text.charAt(i);

        // shift only letters (leave other chars alone)
        if (letter >= 'a' && letter <= 'z') {
            letter = (char) (letter + shift);

            // may need to wrap around
            if (letter > 'z') {
                letter = (char) (letter - 26);
            } else if (letter < 'a') {
                letter = (char) (letter + 26);
            }
        }
        System.out.print(letter);
    }
    System.out.println();
}
```

Self-Test: Strings

Expression	Result?
"This is a string literal."	
"1" + "2"	
1 + " + " + 2 + " = " + 3	
'1' + "2"	
0 + '1' + "2"	
"" + Math.sqrt(2)	
(String) Math.sqrt(2)	
`(string) Math.sqrt(2)`	
"A" == "A"	
"A".equals("A")	
"B" < "A"	
"B".compareTo("A")	
"B".compareTo("B")	
"B".compareTo("C")	

Penn Engineering

Files

Text Files

- Text files are just sequences of characters
- Example file format:

data.txt

```
1  3.4   JSmith
2  3.9   MJones
...
```

- Can read using the **In** library (provided with your textbook)

```
In inStream = new In("data.txt");
while (!inStream.isEmpty()) {
    int idx = inStream.readInt();
    double gpa = inStream.readDouble();
    String name = inStream.readString();
    ...
}
```

Penn Engineering

Functions

Method Overloading

- Two or more methods *in the same class* may also have the same name
- This is called ***method overloading***

absolute value of an `int` *value*

```
public static int abs(int x)
{
   if (x < 0) return -x;
   else       return  x;
}
```

absolute value of a `double` *value*

```
public static double abs(double x)
{
   if (x < 0.0) return -x;
   else         return  x;
}
```

Method Signature

- A method is uniquely identified by
  - its **name** and
  - **its parameter list** (parameter types and their order)
- This is known as its ***signature***

Examples:

```
static    int min (int a, int b)
static double min (double a, double b)
static  float min (float a, float b)
```

Return Type is Not Enough

- Suppose we attempt to create an overloaded `circle(double x, double y, double r)` method by using different return types:

```
static void circle (double x, double y, double r) {...}

//returns true if circle is entirely onscreen, false otherwise
static boolean circle (double x, double y, double r) {...}
```

- This is NOT valid method overloading because the code that calls the function can ignore the return value

```
circle(50, 50, 10);
```

  - The compiler can't tell which `circle()` method to invoke
  - Just because a method returns a value doesn't mean the calling code has to use it

Too Much of a Good Thing

Automatic type promotion and overloading can sometimes interact in ways that confuse the compiler

For example:

```
//version 1
static void printAverage (int a, double b) {
    ...
}

//version 2
static void printAverage (double a, int b) {
    ...
}
```

Why might this be problematic?

Penn Engineering

Too Much of a Good Thing

```
static void printAverage (int a, double b) {/*code*/}
static void printAverage (double a, int b) {/*code*/}
```

- Consider if we do this:

```
public static void main (String[] args) {
     ...
     printAverage(4, 8);
     ...
}
```

- The Java compiler can't decide whether to:
  - promote 7 to 7.0 and invoke the first version of printAverage(), or
  - promote 5 to 5.0 and invoke the second version
- It will throw up its hands and complain
- Take-home lesson: don't be too clever with method overloading

More Documentation

Method-level Documentation

- Method header format:

```
/**
 * Name: circleArea
 * PreCondition: the radius is greater than zero
 * PostCondition: none
 * @param radius - the radius of the circle
 * @return the calculated area of the circle
 */
static double circleArea (double radius) {
    // handle unmet precondition
    if (radius < 0.0) {
        return 0.0;
    } else {
        return Math.PI * radius * radius;
    }
}
```

Method Documentation

- Clear communication with the class user is of paramount importance so that he can
  - use the appropriate method, and
  - use class methods properly.
- Method comments:
  - explain what the method does, and
  - describe how to use the method.
- Two important types of method comments:
  - ***precondition*** comments
  - ***post-conditions*** comments

Preconditions and Postconditions

- Precondition
  - What is assumed to be true when a method is called
  - If any pre-condition is not met, the method may not correctly perform its function.
- Postcondition
  - States what will be true after the method executes (assuming all pre-conditions are met)
  - Describes the side-effect of the method

An Example

Very often the precondition specifies the limits of the parameters and the postcondition says something about the return value.

```
/*Prints the specified date in a long format
      e.g.  1/1/2000 -> January 1, 2000
  Pre-condition:
        1 <= month <= 12
        day appropriate for the month
        1000 <= year <= 9999
  Post-condition:
        Prints the date in long format
*/
static printDate(int month, int day, int year)
{
  // code here
}
```

Function Examples

absolute value of an `int` *value*

```
public static int abs(int x)
{
   if (x < 0) return -x;
   else       return  x;
}
```

absolute value of a `double` *value*

```
public static double abs(double x)
{
   if (x < 0.0) return -x;
   else         return  x;
}
```

overloading

primality test

```
public static boolean isPrime(int N)
{
   if (N < 2) return false;
   for (int i = 2; i <= N/i; i++)
      if (N % i == 0) return false;
   return true;
}
```

multiple arguments

hypotenuse of a right triangle

```
public static double hypotenuse(double a, double b)
{  return Math.sqrt(a*a + b*b);  }
```

Penn Engineering

Function Challenge 1a

Q. What happens when you compile and run the following code?

```
public class Cubes1 {

   public static int cube(int i) {
      int j = i * i * i;
      return j;
   }

   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         System.out.println(i + " " + cube(i));
   }
}
```

```
% javac Cubes1.java
% java Cubes1 6
1 1
2 8
3 27
4 64
5 125
6 216
```

Penn Engineering

Function Challenge 1b

Q. What happens when you compile and run the following code?

```
public class Cubes2 {

   public static int cube(int i) {
      int i = i * i * i;
      return i;
   }

   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         System.out.println(i + " " + cube(i));
   }
}
```

Penn Engineering

Function Challenge 1c

Q. What happens when you compile and run the following code?

```
public class Cubes3 {

   public static int cube(int i) {
      i = i * i * i;
   }


   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         System.out.println(i + " " + cube(i));
   }
}
```

Penn Engineering

Function Challenge 1d

Q. What happens when you compile and run the following code?

```
public class Cubes4 {

   public static int cube(int i) {
      i = i * i * i;
      return i;
   }

   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         System.out.println(i + " " + cube(i));
   }
}
```

Penn Engineering

Function Challenge 1e

Q. What happens when you compile and run the following code?

```
public class Cubes5 {

   public static int cube(int i) {
      return i * i * i;
   }


   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         System.out.println(i + " " + cube(i));
   }
}
```