Methods

Chapter 4: Methods

Asserting Java

Methods

- We have used some existing methods without fully understanding their implementation
  - System.out's **`print`**, **`println`**
  - String's **`length`**, **`charAt`**, **`indexOf`**, **`toUpperCase`**
  - Scanner's **`nextDouble`**, **`nextInt`**
  - BankAccount's **`withdraw`**, **`deposit`**
- Java has thousands of methods
  - We often need to create our own

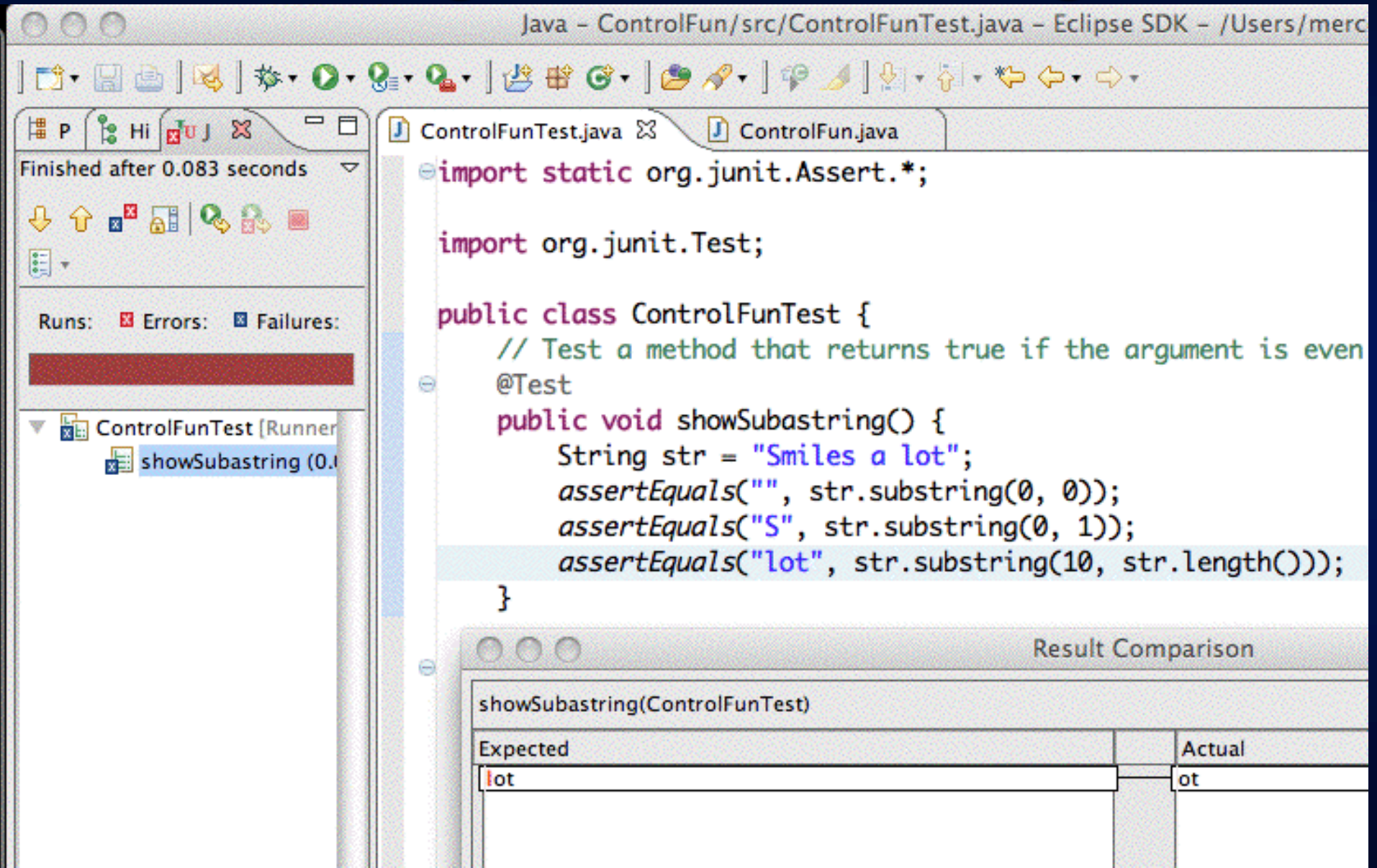
Methods

- There are two major components to a method
  - the method heading with
    - access mode, return type, name, parameters
  - the block
    - a pair of curly braces containing code that fulfills the method's responsibility
- Method headings specify the number and types of arguments required to use the method

Method Heading
with documentation

```
/*
 * Return a new string that is a substring of this string.
 * The substring begins at the specified beginIndex and
 * extends to the character at index endIndex-1.
 * Thus the length of the substring is endIndex-beginIndex.
 *
 * Examples:
 *    "hamburger".substring(4, 8) returns "urge"
 *    "smiles".substring(1, 5) returns "mile"
 *
 * Parameters:
 *    beginIndex - the beginning index, inclusive.
 *    endIndex - the ending index, exclusive.
 *
 * Returns: the specified substring.
 */
public String substring(int beginIndex, int endIndex)
```

Using JUnit to demo substring



What method headings tell us

- Method headings provide the information needed to use it *they show us how to send messages*

```
public String substring(int beginIndex, int endIndex)
    1       2          3      5       4          5       4
```

1. Where is the method accessible
2. What does the method evaluate to?
3. What is the method name?
4. What type arguments are required?
5. How many arguments are required?

Arguments are assigned to parameters

- The substring method requires two arguments in order to specify the portion of the string to return
- When the message is sent
  - the 1st *argument* 0 is assigned to *parameter* **`beginIndex`**
  - the 2nd *argument* 6 is assigned to *parameter* **`endIndex`**

```
fullName.substring(0, 6);
```

```
public String substring(int beginIndex, int endIndex)
```

Implementation of the method is not shown here

Arguments → Parameters

- When a message is sent
  - the first argument is assigned to the first parameter,
  - second argument gets assigned to the second parameter,...
- If you do not supply the correct number and type of arguments, you get compiletime errors

```
fullName.substring("wrong type");
fullName.substring(0, 6, fullName.length());
fullName.substring();
fullName.substring(0.0, 6.0);
```

- BTW: This returns the string form index to the end

```
fullName.substring(2);  // sometimes convenient
```

Method Heading: General Form

- General form of a Java method heading

  `public` *return-type method-name* **`(`** *parameter-1, parameter-2, parameter-n, ...* **`)`**

  - public says a method is known where objects are constructed
  - *return-type* may be any primitive type, any class, or **`void`**
  - A **`void`** method returns nothing, therefore,
    - a **`void`** method can not be assigned to anything
    - a **`void`** method can not be printed with **`println`**

Method Headings

- Example method headings

think of `class` as *type*

```
public char charAt(int index)          // class String
public int indexOf(String sub)         // class String
public void withdraw(double amt)       // class BankAccount
public String getText()                // class Jbutton
public String setText(String str)      // class Jbutton
public void setSize(int x, int y)      // class JFrame
public int nextInt()                   // class Scanner
public int nextDouble()                // class Scanner
public int next()                      // class Scanner
public int nextLine()                  // class Scanner
```

Parameters

- Parameters, which are optional, specify the number and type of arguments required in the message
  - Sometimes methods need extra information
    - How much to deposit?
    - substring need to know begin- and end-indexes?
- General form of a parameter *between (and) in method headings*

class-name identifier

-or-

primitive-type identifier

The Block

- The method body is Java code enclosed within a block `{ }`
- Curly braces have the same things we've seen in `main` methods
  - variable declarations and initializations `int creditOne = 0;`
  - objects `String str = "local";`
  - messages `boolean less = str.compareTo("m") < 0;`
- Method bodies have access to parameters
  - Hence, methods are general enough to be reused with many different arguments

The return statement

- All non-void methods must return a value
  - The type of the value is defined in the method heading
- Use Java's return statement

  `return` *expression* **`;`**

- Example in the context of a method's block

```
public double f(double x) {
  return 2.0 * x - 1.0;
}
```

Code Demo

- Given the following documented method heading,
  - Write a test method in ControlFunTest.java
    - The assertions are expected to fail
  - Write the actual method in ControlFun.java

```
/*
 * Return largest of 3 integer arguments
 * max(1, 2, 3) returns 3
 * max(1, 3, 2) returns 3
 * max(-1, -2, -3) returns -1
 */
public int max(int a, int b, int c) {
  return 0;
}
```

Methods: A Summary

- Method headings provide this information on usage:
  - is the method is available from other places in code?
    - `public` methods are known in the block where constructed
  - *return-type* the kind of value a message evaluates to
  - *method-name* that begins a valid method call
  - *parameter-list* the number and type of needed arguments
  - documentation to describe what the method does
- The block is where your algorithm gets implemented
- The parameters are accessible within the block
- Methods usually return a value of the correct type
- Methods must be fully tested (at least in this course)