Validation and Text Manipulation

ACS-1904 LECTURE 3

Validation and text manipulation

- Debugging a program with the assert statement
- Verifying arguments passed to a method
- Checking field values
- Manipulating text
    - `String: split(), toCharArray()`
    - `StringBuilder`

The assert statement

- The `assert` statement includes a logical expression.
  - If the expression evaluates to `true` then the assert has no effect.
  - But if the expression evaluates to `false` an Assertion Error occurs and the program is terminated.
- Useful to test conditions you expect to be true in your program, and if they are not, then your program terminates immediately and you know what caused the failure

Assert statement

Basic form:

```
    assert logical_expression1;
```

E.g.

```
    assert !line.equals("exit");
```

- If `line` *is equal to* `"exit"` then the program is terminated immediately with an Assertion Error

Assert statement

Alternate form:

```
assert logical_expression1 : expression2;
```

E.g.

```
assert age > 0 : "age must be positive";
```

- If `age` is `<= 0` then the program is terminated immediately with an Assertion Error and the message “age must be positive” is displayed

Assert statement (Ch3ExtraCode.java)

Alternate form:

```
    assert logical_expression1 : expression2;
```

E.g.

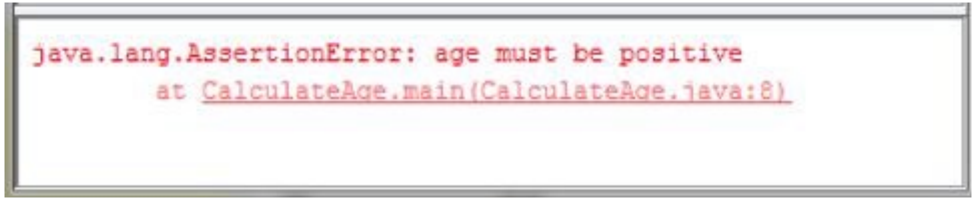
```
assert n > 5 && n < 20 : "n out of range.”;
```

- If n is out of range, i.e. 5 or less or 20 or greater the program is terminated immediately with an Assertion Error and the message “n out of range” is displayed

Example(CalculateAge.java)

Checks the age returned to verify it is positive.

```
public class CalculateAge
{
   public static void main(String[] args){
      int age = getAge("2090-01-01");
      assert age > 0 : "age must be positive";
      System.out.println(age);
}
```



```
java.lang.AssertionError: age must be positive
        at CalculateAge.main(CalculateAge.java:8)
```

Validating parameters

A method expects arguments passed in to its parameters to be appropriate

A robust method checks its parameters to verify they are appropriate.

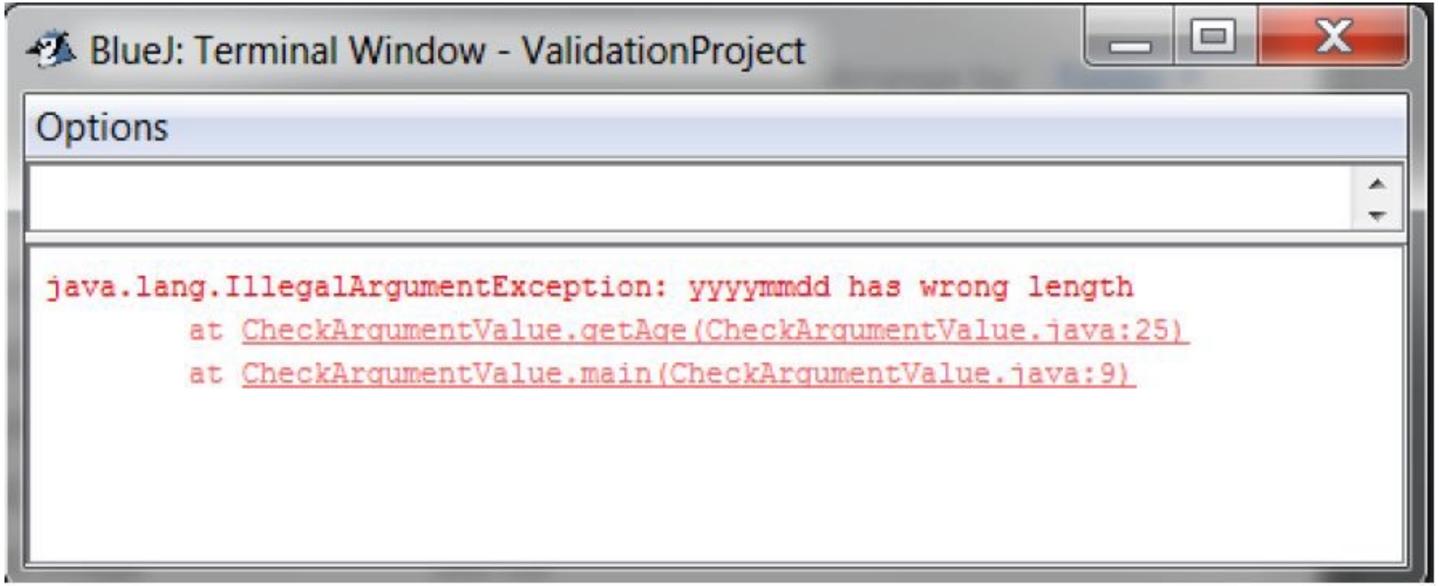
- If a bad argument is found the program can terminate with, say, an Illegal Argument Exception
- If the program fails you know exactly where and why.

Validating parameters(CheckArgumentValue)

```
/**
* getAge - determines age in years
* @param  yyyymmdd   birthdate YYYY-MM-DD
* @return age in years
*/
public static int getAge(String yyyymmdd){
    String arg = yyyymmdd;
    // check length
    boolean valid = arg.length()==10;
    if (!valid)
        throw new IllegalArgumentException("yyyymmdd has wrong length");
        // check for dashes
        valid = arg.charAt(4)=='-' && arg.charAt(7)=='-' ;
        if (!valid)
           throw new IllegalArgumentException("yyyymmdd does not have
dashes in correct places");
…
```

Validating parameters

When an illegal argument exception occurs:|



```
java.lang.IllegalArgumentException: yyyymmdd has wrong length
        at CheckArgumentValue.getAge(CheckArgumentValue.java:25)
        at CheckArgumentValue.main(CheckArgumentValue.java:9)
```

Manipulating text - toCharArray method

toCharArray() – a string of characters is converted to an array of char

`"abc".toCharArray()` generates the char array `{'a','b','c'}`

Instead of coding

```
for (int i=0; i<line.length(); i++)
    if (Character.isDigit(line.charAt(i))) …
```

You can use

```
char[] myLine = line.toCharArray();
for (char c: myLine)
    if (Character.isDigit(c)) …
```

Manipulating text - `split` method

`split()` – a string of characters is split into an array of strings based on a *regular* expression

E.g.

```
"John A. McDonald".split(" ");
```

generates the String array of 3 elements:

```
{"John", "A.", "McDonald"}
```

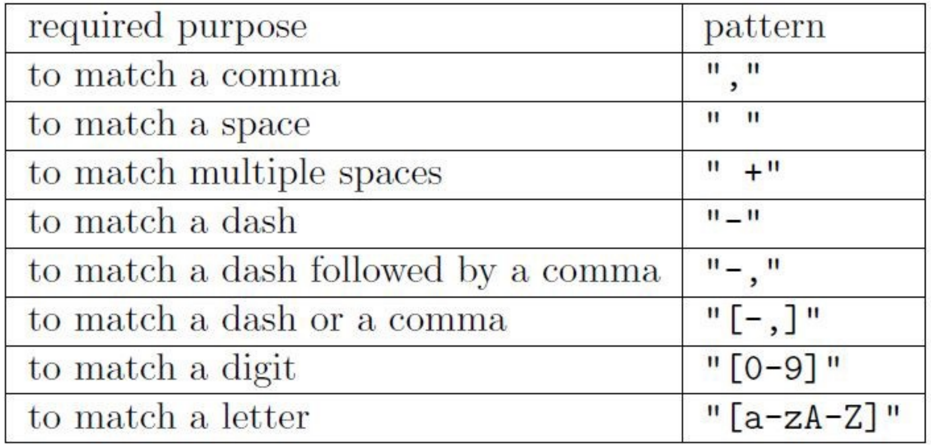
E.g.

```
String course = "ACS-1904-003 ";
String[] parts = course.split ("-");
```

```
parts[0] is "ACS"
parts[1] is "1904"
parts[2] is "003"
```

Manipulating text - `split` method

regular expression – a pattern for searching purposes



required purpose	pattern
to match a comma	`","`
to match a space	`" "`
to match multiple spaces	`" +"`
to match a dash	`"-"`
to match a dash followed by a comma	`"-,"`
to match a dash or a comma	`"[-,]"`
to match a digit	`"[0-9]"`
to match a letter	`"[a-zA-Z]"`

Note: [and] specify any character within the brackets,
+ specifies one or more of a preceding character,
- specifies a range of characters

Manipulating text - `split` method

Special characters:

```
. + * ? ^  $  (  ) [  ]  {  }  |  \
```

To search for any of these characters you have to "escape" them, but wait isn't the '\' the escape character? Yup. So….

.split("\\+"); will split a string using the + as a delimiting character

Manipulating text - `split` method

Examples in text:

E.g. 3(CatenateWords.java)

- A string of words separated by one or more spaces is split into its separate words, and then recombined capitalizing the first character of each word.
  - Uses `for(String s : line.split(" +"))`

E.g. 4(ValidateFormat.java)

- Validates a SIN where groups of 3 digits are expected to be separated by a dash.
  - Uses `sin.split("-");`

Manipulating text - StringBuilder

A text string that is of type `String` is ***immutable***

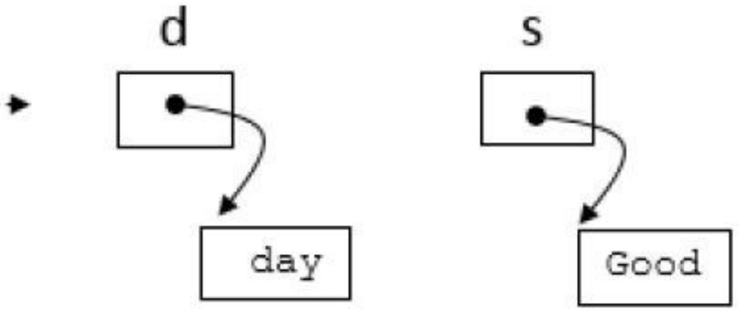
- *Immutable* – once initialized, it cannot be changed.
- But we can *change* the value of a String variable
  - any time the value of string variable *changes* it is actually allocated a new area in memory

Manipulating text - StringBuilder

Figure 3.3

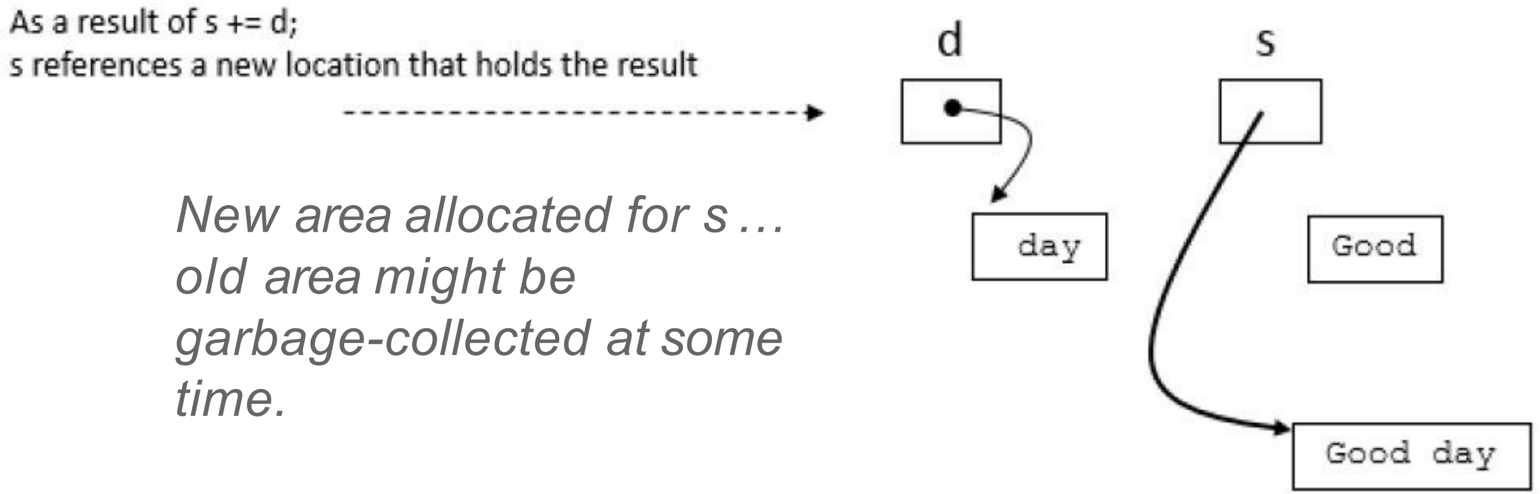
When these two instructions execute

```
String s = "Good";
String d = " day";
We have d and s
```



Manipulating text - StringBuilder

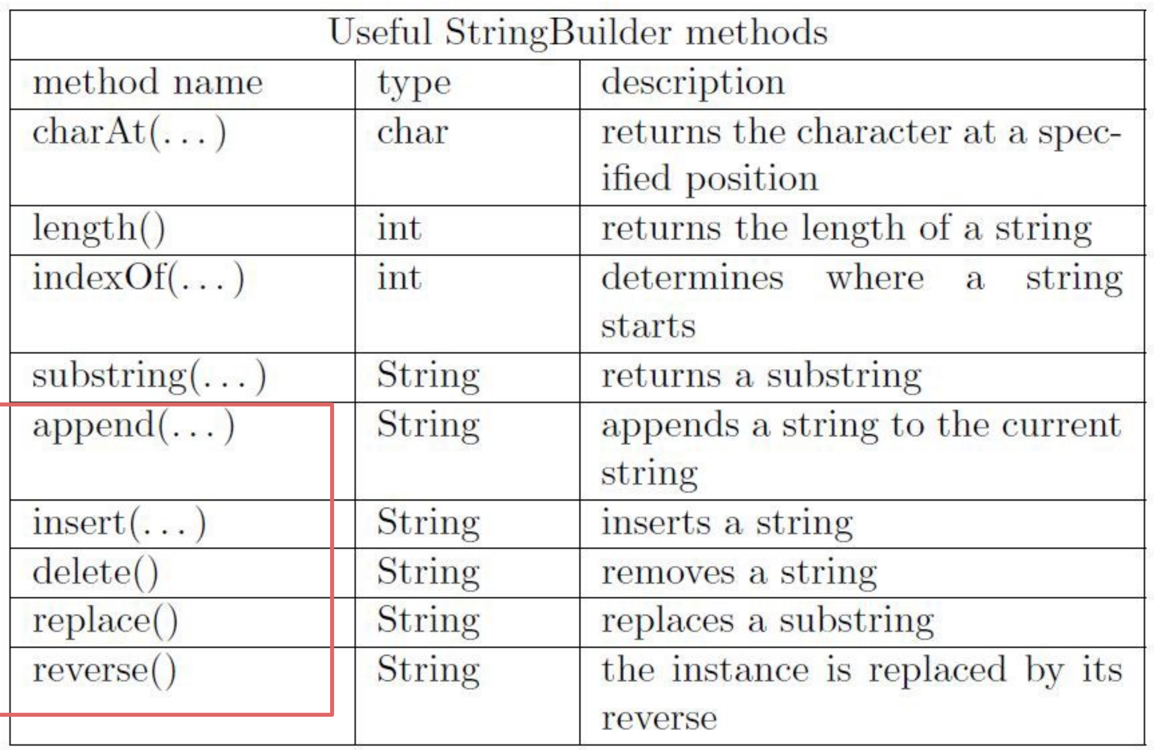
Figure 3.3



New area allocated for s … old area might be garbage-collected at some time.

Excessive text manipulation can be expensive in terms of time, and so `StringBuilder` can be used

Manipulating text - StringBuilder



Useful StringBuilder methods		
method name	type	description
charAt(. . .)	char	returns the character at a specified position
length()	int	returns the length of a string
indexOf(. . .)	int	determines where a string starts
substring(. . .)	String	returns a substring
append(. . .)	String	appends a string to the current string
insert(. . .)	String	inserts a string
delete()	String	removes a string
replace()	String	replaces a substring
reverse()	String	the instance is replaced by its reverse

Not available with String - only StringBuilder

StringBuilder’s append method(CatenateStringsWithStringBuilder.java)

```java
import java.util.Scanner;
import java.io.File;
import java.io.IOException;
public class CatenateStringsWithStringBuilder
{
    public static void main(String[] args) throws IOException {
        Scanner f = new Scanner(new File("ReadMe.txt"));

        StringBuilder result = new StringBuilder(1000);
        while (f.hasNext()){
            result.append(f.next());
        }
        System.out.println(result);
    }
}
```

StringBuilder’s reverse method(ReverseString.java)

```
StringBuilder original = new StringBuilder(kb.next());
StringBuilder reversed = new StringBuilder(original);
// reverse one of these
reversed.reverse();
System.out.println("original : " +original.toString());
System.out.println("reversed : " +reversed.toString());
// test for equality
// need to compare strings
// because StringBuilder does not
// override equals in Object
if (original.toString().equals(reversed.toString()))
    System.out.println("a palindrome");
else
    System.out.println("not a palindrome");
```