# Java Basics, Strings

**Kavita Ganesan, Ph.D.**

www.kavita-ganesan.com



## Lecture Outline

- Intro to Strings + Creating String Objects •

String Methods + Concatenation

- String Immutability + StringBuilder
- String Equality/Inequality
- Wrapper Classes
- Review + Quiz
- Download code samples + take home exercise



# STRINGS - INTRO, CREATION

## What is a String?

**String**  a sequence of characters

Example of **strings:**

"The cow jumps over the moon" "PRESIDENT OBAMA" "12345"

# What is a **String class** in Java?

- String class in Java: holds a "sequence of characters"

String **greeting** = new String("Hello

world!“);

- Creating new String object
- Contains sequence of chars

# String Class

- Why use String class vs. char array?
- **Advantage:** provides many useful methods for string manipulation
  - Print length of string – **str.length()**
  - Convert to lowercase – **str.toLowerCase()**
  - Convert to uppercase – **str.toUpperCase()**

* Many others



# Creating String Objects

There are 2 ways to create String objects

**Method 1:**

```
String greeting1 = new String("Hello World!") ;
```

String **greeting1** = **new** String("Hello World!") ; • **new** operator for

creating

- **Variable** of **type String**
- **String value** aka *string literal*

instance of the **String** class

- **Name** of variable is **greeting1**
- **String literal:** series of

- **Recall:** Instance of a class is an object

in double quotes.

characters enclosed

# Creating String Objects

There are 2 ways to create String objects

**Method 2:**

String **greeting2** = "Hello World Again!"

String **greeting2** = “Hello World Again!” ;

- **Shorthand** for String creation (most used)
- **Behind the scenes:** new instance of String class with “Hello World Again!” as the value

# Creating String Objects

There are 2 ways to create String objects

**Method 1:**

String **greeting2** = “Hello World Again!” ;

String **greeting1** = **new** String(“Hello World!”) ;

String

**greeting1** = **new** String(“Hello World!”) ; **Method 2:**

String

**greeting2** = “Hello World Again!” ;

**Local Variable Table String Objects**

**greeting1**

*Holds a reference*

*Holds a reference*

“Hello World!”

**greeting2** “Hello World Again!”

# String Constructor

- **Recall:** when new object created  the constructor method is always called first

- Pass initial arguments or empty object

- String class has multiple constructors



# String Constructor

String **str2**= new String(**“string”**) ; //string input

String **str1**= new String() ; //empty object

String **str1**= new String() ; //empty object

String **str2**= new String(**“string”**) ; //string input

String **str3**= new String(**char[]**) ; //char array input

String **str4**= new String(**byte[]**) ; //byte array input*

few others



# Strings: Defining and initializing

## Simple example

String **s1** = "Welcome to Java!";

```
String s2 = new String("Welcome to Java!"); //same as s1
```

**Numbers as strings**

```
String s3 =“12345”;
String s4 = new String(s3); //s4 will hold same value as s3
```

**Char array as strings**

```
char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.' };
String s5= new String(helloArray);
```



# Strings: Defining and

# initializing Empty Strings

String **s5** = "";

String **s6** = new String(“”);

String object created; String value is empty “”

String **s7** = null;

String variable not pointing to any String object

## Null String

# NetBeans Examples…

# Understanding String Creation

String **greeting1** = "Hello Utah!"

String **greeting2** = "Hello Utah!"

**Does Java create 2 String objects internally?**

Without **"new"** operator for String creation:

- Java looks into a String pool (collection of String objects)
  - Try to find objects with same **string** value
- If **object exists** → new variable points to existing object •

If **object does not exist** → new object is created •

Efficiency reasons – to limit object creation



# Understanding String Creation

String **greeting1** = “Hello Utah!”
String **greeting2** = “Hello Utah!”

**Pool of String Objects**

“Hello Utah!”

**Local Variable Table**

greeting1

**Concept of String pooling**

greeting2

# Understanding String Creation

⚠

String **greeting1** = **new** String (“Hello Utah!”);
String **greeting2** = **new** String (“Hello Utah!”);

**String Objects**

**Local Variable**

“Hello Utah!”

**Table** greeting1

“Hello Utah!”

greeting2

# NetBeans Examples…

# STRINGS - METHODS,

# CONCATENATION
# String Methods

**Advantage of String class:** many built-in methods for String manipulation

```
str.length(); // get length of string
str.toLowerCase() // convert to lower case
str.toUpperCase() // convert to upper case
str.charAt(i) // what is at character i?
str.contains(..) // String contains another string?
str.startsWith(..) // String starts with some prefix?
str.indexOf(..) // what is the position of a character?
….many more
```

# String Methods - length, charAt

str.length() → Returns the **number of chars** in String

str.charAt(i) → Returns the **character at position** i

Character positions in strings are numbered starting from 0 – just like arrays

String Methods - length, charAt

str.length() → Returns the **number of chars** in String

str.charAt(i) → Returns the **character at position** i

a

**"Utah".**length();

**"Utah".**charAt (2); 0123

**Returns**: 4

# String Methods – valueOf(X)

String.valueOf(**X**) - Returns **String representation** of

**X** • ***X:*** ***char, int, char array, double, float, Object*** •

Useful for converting different data types into String

String **str1** = String.**valueOf**(**4**); //returns “4”

String **str2** = String.**valueOf**(**‘A’**); //returns “A” String **str3** = String.**valueOf**(**40.02**); //returns “40.02”

# String Methods – substring(..)

**str.substring(..)**  returns a new String by copying characters from an existing String.

- str.**substring** (**i**, **k**)

– returns substring of chars from pos i to k-1

- str.**substring** (**i**);

– returns substring from the i-th char to the end 

# String Methods – substring(..)

"Tom".substring(**9**);

012

**Returns**:

**"Be"**

"Ben".substring(**0**,**2**);

012

"John".substring(**1**);

0123

**"ohn"**

“” **(empty)**



# String Concatenation – Combine Strings

- What if we wanted to combine String values?

  String **word1** = “re”;

  String **word2** = “think”;

  String **word3** = “ing”;

How to combine and make  **“rethinking” ?** •

Different ways to concatenate Strings in Java



# String Concatenation – Combine Strings

String **word1** = “re”;
String **word2** = “think”;
String **word3** = “ing”;

**Method 1:** Plus “+” operator

String **str** = word1 **+** word2;

– *concatenates word1 and word2* ⇒ ***"rethink"***

**Method 2:** Use String's "concat" method String **str** = word1.**concat** (word2);

– *the same as word1 + word2* ⇒ ***"rethink"***



# String Concatenation – Combine Strings

Now **str** has value **"rethink",** how to make **"rethinking"**? String **word3** = "ing";

**Method 1:** Plus "+" operator

**str = str +** word3; //results in ***"rethinking"***

**Method 2:** Use String's "concat" method **str = str.concat**(word3); //results in ***"rethinking"***

**Method 3:** Shorthand

**str +=** word3; //results in ***"rethinking"*** (same as method

1) 28

# String Concatenation: Strings, Numbers & Characters

String **myWord**= "Rethinking";

int **myInt=**2;

char **myChar**='!';

**Internally:** myInt & myChar converted to

operator

String objects

## **Method 1:** Plus "+"

String **result** = myWord **+** myInt **+** myChar;

*//Results in "Rethinking2!"*

# String Concatenation: Strings, Numbers & Characters

```
String myWord= “Rethinking”;
int myInt=2;
char myChar=‘!’;
```

**Method 2:** Use String’s “concat” method

```
String strMyInt= String.valueOf(myInt);
String strMyChar=String.valueOf(myChar);
```

```
String result =
myWord.concat(strMyInt).concat(strMyChar); //Results in
"Rethinking2!"
```

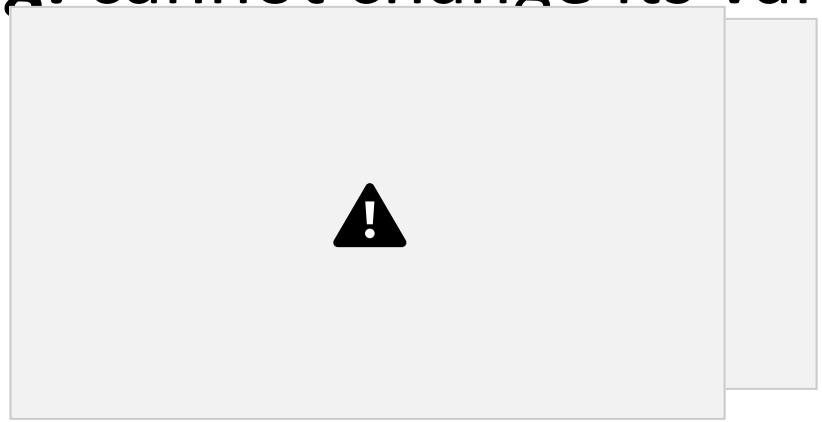


# NetBeans Examples…

# STRING IMMUTABILITY

# String Immutability

- Strings in Java are immutable
- **Meaning:** cannot change its value, once created



String **str;**

**str** = “Java is Fun!”; **str** = “I hate Java!”;

Did we change the value of **“Java is Fun!”** to **“I hate**

**Java!”**?



# String Immutability

= “I hate Java!”;

Cannot insert or remove a value

String **str;**

**str** = “Java is Fun!”; **str**

**Local Variable Table**

**str**

**String Objects**

"Java is Fun!"

"I hate Java!"

***e.g. remove "Fun"***



# String Immutability

**str;**

**str** = “Re”;

**str** = str + “think”;

//Rethink

**Every concat:** Create new String object

**Unused objects:** “Re”, “Rethink” go to garb. coll.

**str** = str + “ing”; // Rethinking

**Local Variable Table** “Rethink”

**str** “Rethinking”

**String Objects** “Re”



# String Immutability

- **Problem:** With frequent modifications of Strings
  - Create many new objects – uses up memory – Destroy many unused ones – increase JVM workload – **Overall:** can slow down performance

- **Solution** for frequently changing Strings:
  - StringBuilder Class instead of String



# StringBuilder Class

- StringBuilders used for String concatenation

- StringBuilder class is "**mutable**"
  **Meaning:** values within StringBuilder can be changed or modified as needed

- In contrast to Strings where Strings are immutable



# StringBuilder - "append" method

- append(X)  most used method
  - Appends a value **X** to end of StringBuilder – ***X: int, char, String, double, object (almost anything)***



# StringBuilder for String Construction

```
StringBuilder sb = new StringBuilder(); //obj creation
sb.append("Re"); //add "Re" to sb
sb.append("think"); //add "think" to sb
sb.append("ing"); //add "ing" to sb

String str= sb.toString(); //return String value
```

- **Only 1 object**
- **All 3 words appended to same object**

**Local Variable Table StringBuilder Object**

sb

Rethink ing

***Bottom-line:*** *Use StringBuilder when you have frequent string modification*



# STRING EQUALITY/INEQUALITY



## Testing String Equality

- How to check if two Strings **contain same value**?

eferencing

```
if(str1==str2) { //eval to false
System.out.println("same")
; }
```

same object as **str2**?

| **Local Variable Table** | **String Objects** |
| --- | --- |
| **str1** | "Hello World!" |
| **str2** | "Hello World!" |

# Testing String Equality

- How to check if two Strings **contain same value**?

String

**str1=**new

String(“Hello World!”); String

```
str2=new String("Hello World!");

if(str1==str2) { //eval to false
System.out.println("same" same  as str2?
); }
if  content  of str1


if(str1.equals(str2)) { //eval to
true
System.out.println("same"); }
```



# Testing String Equality • What

# if “new” operator not used?

String **str1** = “Hello World!”; String **str2** = “Hello World!”;

if(str1==str2) { //eval to true
System.out.println(“same”)
; }

if **str1 referencing** same object as **str2**?

**String Objects**

**Local Variable Table**

**str1** “Hello World!” **str2**



# Testing String Equality • What

if “new” operator not used?

```
String str1 = “Hello World!”; String str2 = “Hello World!”;

if(str1==str2) { //eval to true
System.out.println(“same”); }

if(str1.equals(str2)) { //eval to true System.out.println(“same”);
```

```
}
```



# Testing String Equality

- **Point to note:** String variables are references to String objects (i.e. memory addresses)

- **"str1==str2"** on String objects compares memory addresses, not the contents

- Always use **"str1.equals(str2)"** to compare contents

# NetBeans Examples…

# WRAPPER CLASSES

# (SIDE TOPIC)



## Wrapper Classes

- **Recall:** primitive data types int, double, long are not objects

- **Wrapper classes:** convert primitive types into objects
  - **int:** Integer wrapper class
  - **double:** Double wrapper class
  - **char:** Character wrapper class

- 8 Wrapper classes:

– Boolean, Byte, Character, Double, Float, Integer, Long, Short

# Wrapper Classes

- Why are they nice to have?
  - Primitives have a limited set of in-built operations – Wrapper classes provide many common functions to work with primitives efficiently
- How to convert a **String "22"** → **int**?
  - Cannot do this with primitive type int
  - Can use Integer wrapper class: Integer.parseInt(..)

String **myStr** = "22";
int **myInt**=
Integer.parseInt(myStr);

# Wrapper Classes

- **Reverse:** How to convert **int 22  String**?
  - Cannot do this with primitive int
  - Can use Integer wrapper class: Integer.toString(..)

int

```
myInt= 22;
String myStr= Integer.toString(myInt); // “22”
String myStr2 = String.valueOf(myInt); //
```

## “22” **equivalent**

- Each Wrapper class has its own set of methods

# Integer Wrapper Class Methods

# Character Wrapper Class Methods

# Double Wrapper Class Methods

# Review + Quiz

# Review + Quiz (1)

- What is the purpose of a String class? –
  *String class holds a sequence of characters*
- Why use a String class vs. a char array?
  – *String class has many built-in methods*
  – *Makes string manipulation easy*
- What are the 2 ways to create Strings? –
  *new operator;* *String str=new String("Hello World");* –
  *direct assignment;* *String str="Hello World";*



# Review + Quiz (2)

- Can we directly modify String objects?
  - *No, Strings are immutable by design*
  - *Once created **cannot be changed** in any way*
  - *Only the variable **references** are **changed***

- For frequent modification of Strings should we use the String class?
  - *No, use **StringBuilder** class instead of String class*
  - *String class creates many new objects for concatenation – slows things down*



# Review + Quiz (3)

- What are the different ways to concatenate Strings?
  - *- Use plus **"+"** operator or in-built **"concat"** method*
- When you use "==" when comparing 2 String objects, what are you comparing?
  - *– comparing references (address in memory)*
- How do you compare contents of two String objects?
  - *– string1.equals(string2)*



# Review + Quiz (4)

- What are Wrapper classes and why do we need them?
  - *convert primitive types into objects*
  - *provide many functions to work with primitives efficiently*

- What method do you use to convert a primitive **int** or **double** into a String representation?
  - *Double.toString(doubleVal), Integer.toString(intVal)*
  - *String.valueOf(primitiveType)*



# End of Quiz!

# Take-Home Exercise & Code Samples

- Go to https://bitbucket.org/kganes2/is4415/downloads
- Download StringNetBeans.zip
- Import Code into NetBeans

  1. Select **File** -> **Import Project** -> **From Zip**
  2. Select downloaded StringNetBeans.zip 3.

  You should see TestProject in NetBeans

- Complete all tasks in **StringExercise.java** •

Code samples in "**examples**" package

Example code run in

class Take-home

exercise

StringExercise.java
Take-home
exercise



# Further Questions…

**Kavita Ganesan**

ganesan.kavita@gmail.com

www.kavita-ganesan.com



# Online videos

- https://www.youtube.com/results?search_query=java+strings