Advanced Programming Techniques in Java

Review: Data Types

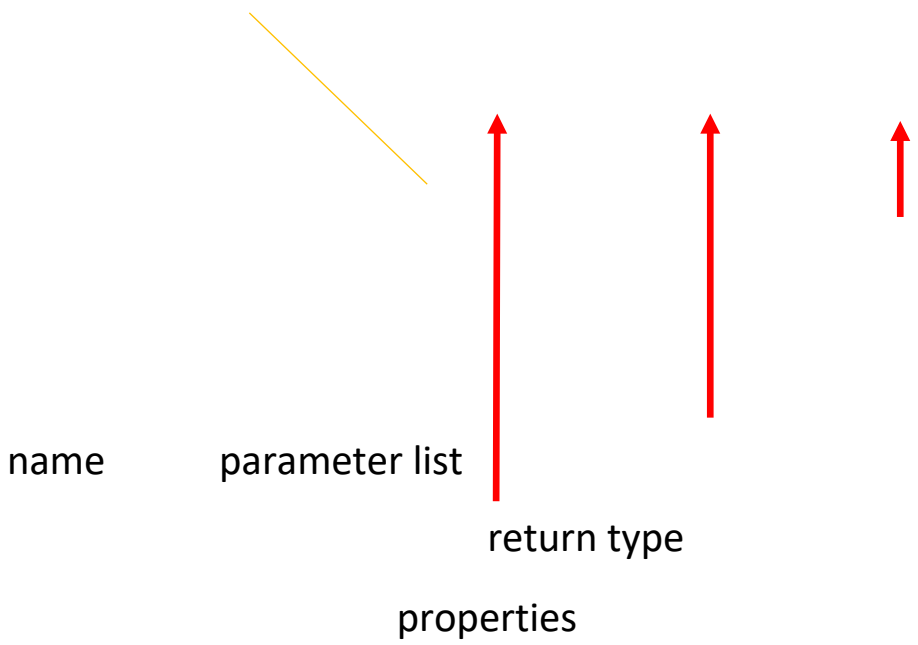
- Data types classify the different values t variable
- In Java there are two types of

- Primitive Data Types
- Non-primitive Data Types

Review: Methods

- A method declaration begins with a *method header*

```
public static  int    add   ( int num1
```



- The parameter list specifies the type and name of each p
- The name of a parameter in the method declaration is *argument*
- **static** indicates a *static* or an *object/instance* method
- A method that is not static, is an instance method

Review: P

- A *parameter* is a special type of variable that a
  a method
- A method can accept multiple parameters (se
- Each time a method is called, the *actual parar*
  copied into the formal

Declaration syntax

```
public static  int    add   ( int num1, in
```

Call syntax

formal para

```
add        (5, 9);
```

actual parameters

To summa

```
public class AddingTwoNumbers{

    public static void main (String[] args){
        int x = add (2, 3); // method call
        System.out.println(x);
    }

    public static int add (int num1, int num2){
            int result = 0;
            result = num1 + num2;
            return result;
    }

}
```

`main` Me

- The `main` method is a special method with a
  execution of a program starts

```
public static void main(Str
    // method body
}
```

Class obj

- Strings
- Primitive and Reference (Non-Primitive) types
- Randomization

7

String

Data Type

- Data types classify the different values to be
  there are two types of data types:

- Primitive Data Types
- Non-primitive Data Types

Strings

- "Hello, world!" or "Enter a number
- Java supplies a class called String used to

- A *string* is an object storing a sequence of cha

- String objects have
- Fields (or data values): the characters in the string
- Methods (or operations): get the length of the string, g

- Strings in Java are immutable, which means th value can never change
- "Hello, world!" or "Enter a number

Strings

- Java supplies a class called `String` used to
  We'll first learn how to use objects and later how to
- A *string* is an object storing a sequence of cha
- Objects have
- Fields (or data values). For strings the fields are the cha
- Methods (or operations). For strings some of the opera
  substring, etc.
- Unlike most other objects, a `String` does n
  using the keyword `new`

Strings

- You can simply declare variable of type `Stri` special syntax that works only for strings)

```
String s1 = "hello";
String s2 = "there";
```

- When declaring a string you can have a value

```
String combined = s1 + " " + s2;
```

- Remember the **+** symbol concatenates string `there"` ▪ The `String` class provides ma manipulate strings.
- Many of the methods return a value, such as an intege

Strings

- Once an object has been instantiated, we can
methods

- Given:

```
String s1 = "hello";
String s2 = "how are you?";
```

- To know how many characters are in `s1` and
`length()` method

```
System.out.println s1.length() ("Length o
System.out.println s2.length() ("Length o
```

Strings

- The `length()` method returns the length of
  are in the string)
- How to get the individual characters by thems

Strings

- The characters of a string are numbered with

```
String name = "R. Kelly";
```

index	0	1	2	3	4
character	R	.		K	e

- First character's index : 0
- Last character's index : 1 less than the string's length
- `char` **`charAt(index)`** method returns th

  string

```
name.charAt(0) 'R' name.charAt(3) 'K'
```

- How to get a substring?

Strings

`String s2 = "How are you?"` (e.g. the subs

index	0	1	2	3	4
character	H	o	w		a

- The method **`substring(start, end)`** re
characters as the substring that begins at inde
`end`

```
s2.substring(8,12);  "you?"
s2.substring(4,7); "are"
```

Strings

Method name	
`charAt(`**`index`**`)`	Returns the ch string
`length()`	Returns the nu
`substring(`**`index1, index2`**`)` or `substring(`**`index1`**`)`	Returns the ch (inclusive) to **i** **index2** is om
`toLowerCase()`	Returns a new
`toUpperCase()`	Returns a new
…	

`String` example

```
public class Test { public static void
   main(String[] args) {
      String s1 = "hello";
      String s2 = "class";
      String s3 = "soon we'll have the firs

      System.out.println(s1 + " " + s2 + " 
      System.out.println();

      //Use of the method length()
      String s4 = s1 + " " + s2 + " " + s3;
      int strLen = s4.length ();
      System.out.println("The length of the
      System.out.println();

      //Use of the method charAt(index)
      char ch = s4.charAt (3);
      System.out.println("The character in 
      System.out.println();

      //Use of the method toUpperCase()
      String newString = s4.toUpperCase();
```

```
        System.out.println(newString);
    }
}
```

`char` vs.

- `char` is a primitive type representing a sing
- A string is an object. E.g., `"h"`

```
String s = "h";
s = s.toUpperCase();        // "H" i
s.length();        //  1 char first =
s.charAt(0);   // 'H'
```

- A `char` is a primitive data type; you car

Comparing ch values

- You can compare `char` values with ==,

```
String word = console.next();
char last = word.charAt(word.le
if (last == 's') {
      System.out.println(word +
}
```

```
for (char c = 'a'; c <= 'z'; c+
System.out.print(c);
}
```

`char` vs.

- Each `char` is mapped to an integer value

  `'A'` is 65 `'B'` is 66 `''` is 32

  `'a'` is 97 `'b'` is 98 `'*'` is 42

- Mixing a `char` and an `int` causes autom

  `'a'` + 10 is 107

- To convert an `int` into the equivalent `ch`

  `(char) ('a' + 2)` is `'c'`

ASCII table

Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space
1	01	Start of heading	33	21	!
2	02	Start of text	34	22	"
3	03	End of text	35	23	#
4	04	End of transmit	36	24	$
5	05	Enquiry	37	25	%
6	06	Acknowledge	38	26	&
7	07	Audible bell	39	27	'
8	08	Backspace	40	28	(
9	09	Horizontal tab	41	29	)
10	0A	Line feed	42	2A	*
11	0B	Vertical tab	43	2B	+
12	0C	Form feed	44	2C	,
13	0D	Carriage return	45	2D	-
14	0E	Shift out	46	2E	.
15	0F	Shift in	47	2F	/
16	10	Data link escape	48	30	0
17	11	Device control 1	49	31	1
18	12	Device control 2	50	32	2
19	13	Device control 3	51	33	3
20	14	Device control 4	52	34	4
21	15	Neg. acknowledge	53	35	5
22	16	Synchronous idle	54	36	6
23	17	End trans. block	55	37	7
24	18	Cancel	56	38	8
25	19	End of medium	57	39	9
26	1A	Substitution	58	3A	:
27	1B	Escape	59	3B	;
28	1C	File separator	60	3C	<
29	1D	Group separator	61	3D	=
30	1E	Record separator	62	3E	>
31	1F	Unit separator	63	3F	?

Comparin

- Relational operators such as < and ==

```
Scanner console = new Scanner(Syst
System.out.print("What is your nam
"); String name = console.next();
(name == "Barney") {
   System.out.println("I love you,
   System.out.println("We're a hap
}
```

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

Data Type

- Data types classify the different values to b

  there are two types of data types:

- Primitive Data Types
- Non-primitive Data Types

The equa

- Objects are compared using a method nar

```
Scanner console = new Scanner(
System.out.print("What is your
"); String name = console.next
(name.equals("Barney")) {
   System.out.println("I love
   System.out.println("We're a
}
```

- This is a method that returns a value of t logical tests

Primitive and Ref

Prim

- Some types of data are stored inside their vari

```
int x = 7;
```

x 7

- These data types are known as primitive types:
  - `int`
  - `long`
  - `double`

```
boolean
char
```

Object vs. Primi

- If something is not primitive, it is an object
- An object is a construct that groups together:
  - one or more data values (the object's *attributes* or *
  - one or more *methods*

- Every object is referred as an **inst**

Reference Types

- Objects are stored as reference
  - The object is stored outside the variable
  - The variable stores a reference (memory a

```
String s1 =
"hello, world"; s1
```

```
"hello, world"
```

- Data types that work this way are known as re
- Variable of those types are reference variables

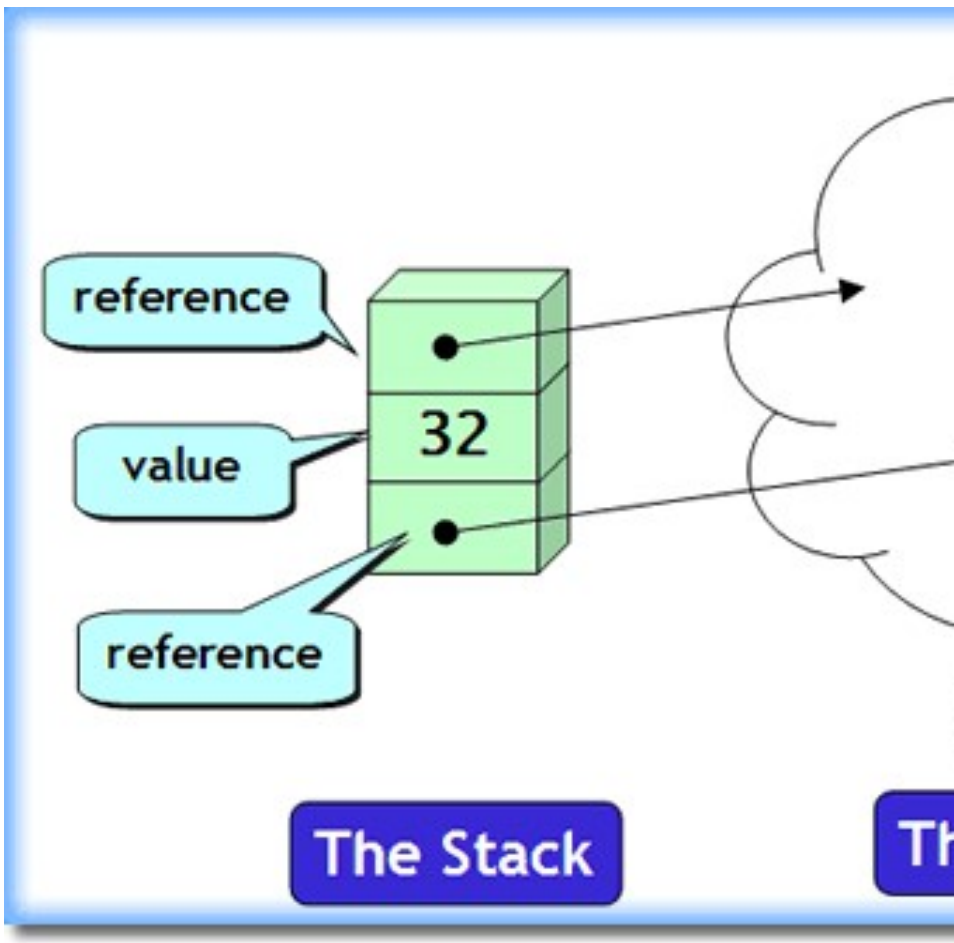
Why Declare Va

- Different primitive values require different am

- When declaring a variable, we
  memory to allocate
  - `int` (4 byte
  - …

- In Python everything is an object, thus all varia

Memory Model



Value se

- Value semantics (or value types): behavio
assigned, passed as parameters, or returne
use value semantics
- When one variable is assigned to another

Modifying the value of one variable does no

```
int x = 5; int y = x;      // x
= 5, y = 5 y = 17;         // x
= 5, y = 17 x = 8;          //
x = 8, y = 17
```

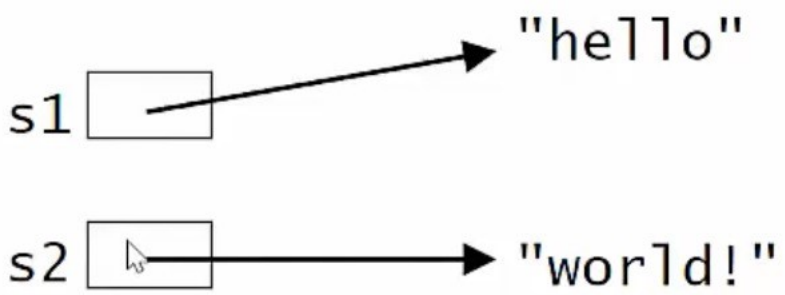
Referenc

- If a variable represents an object, the obj
  variable
- The object is located somewhere else in
  memory address of the object
  - We say that the variable stores a reference to
  - Such variables are called reference variables
- When one variable is assigned to anothe
  variables refer to the same object
- Modifying the value of one variable will a

The `String` ob

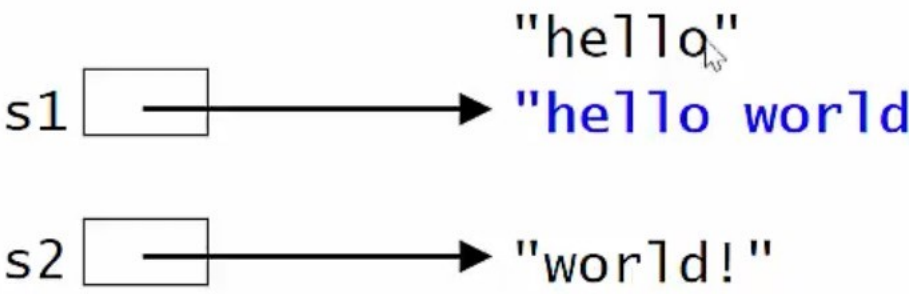
The `String` ob

```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length
s2 = s1.substring(0, 5)
      + s1.charAt(numCha
String s3 = s2.toUpperCa
```



The `String` ob

```
String s1 = "hel
String s2 = "wor
s1 = s1 + " " +
int numChars = s
s2 = s1.substrin
      + s1.charA
String s3 = s2.t
```

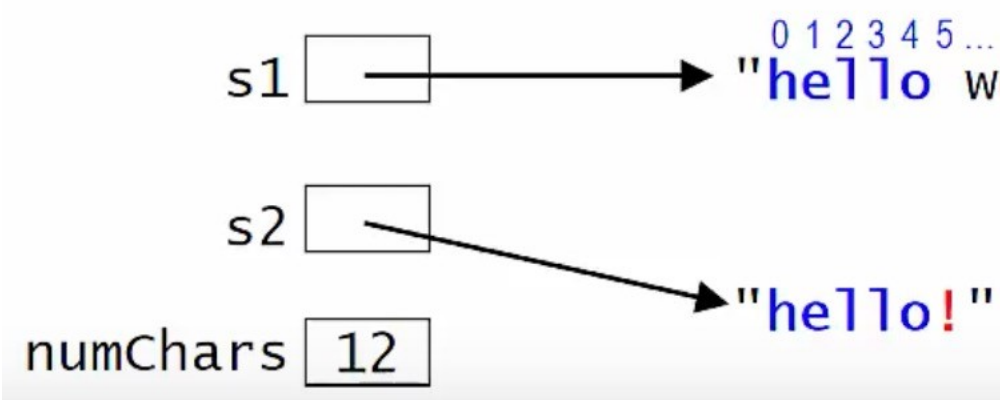


Strings are immutable

The `String` ob

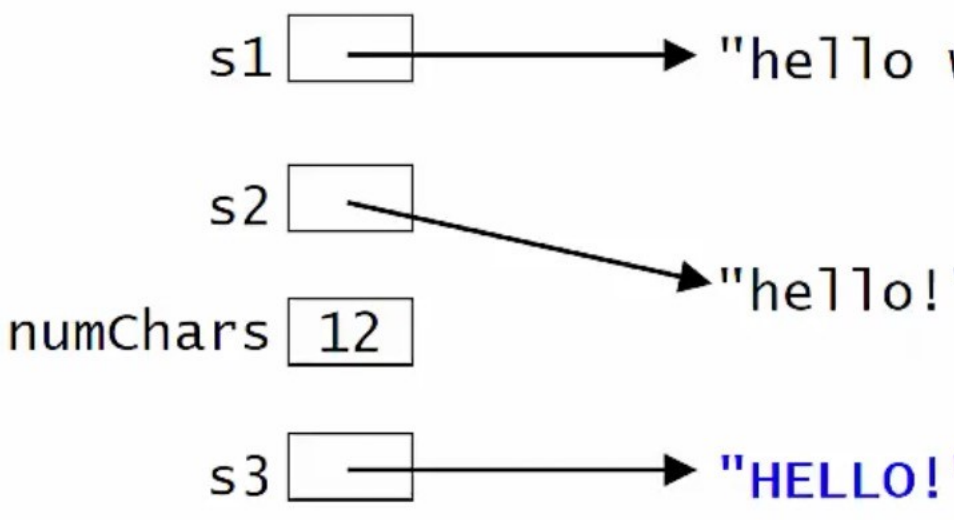
The `String` ob

```
String s1 =
String s2 =
s1 = s1 + "
int numChars
s2 = s1.subs
      + s1.c
String s3 =
```



The `String` ob

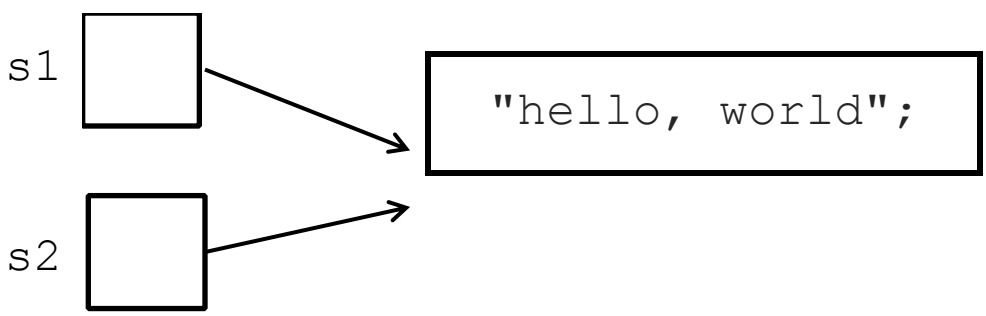
```
String s1 =
String s2 =
s1 = s1 + "
int numChar
s2 = s1.sub
      + s1.
String s3 =
```



Copying

- When we assign the value of one referen
copy the reference to the object We do n

- ```
  String s1 = "hello, world";
  String s2 = s1;
  ```


  ```

`null` ref

- To indicate that a reference variable doe
  can assign it a special value called `null`

```
String s = null;
```

- Attempting to use a `null` reference to a
  `NullPointerException`
  - Pointer is another name for reference

```
char ch = s.charAt(5); //NullPointerE
```