HBO Graduaat Informatica Optie Programmeren

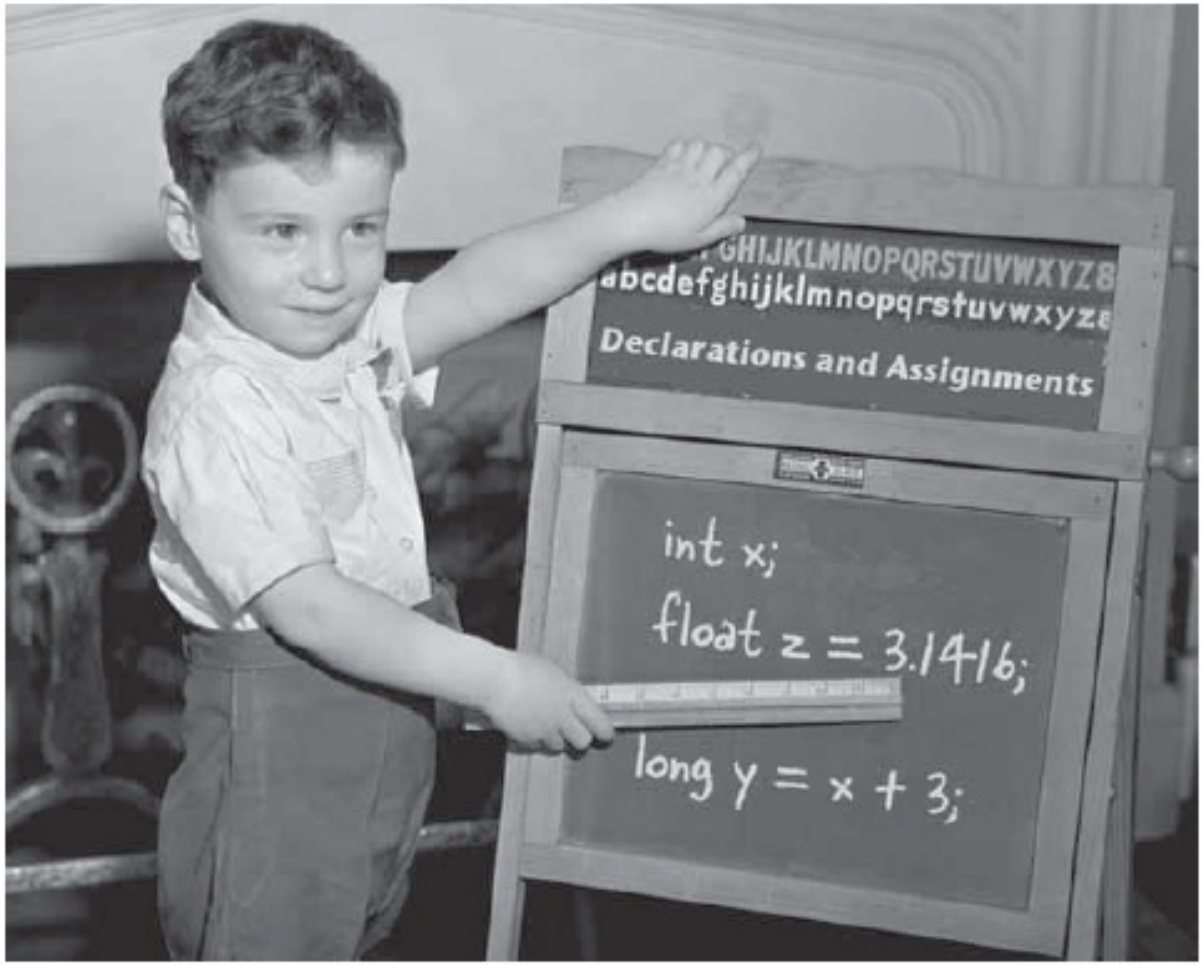
Java Basics

Variables, literals & Operators

cvo leerstad

volwassenenonderwijs

Know Your Variables



Variables, Literals and Operators

- Types and Variables
  - 2 types
    - primitive types
      - boolean
      - numeric types
        - » integral : byte, short, int, long, char
        - » floating-point : float, double
    - reference types
      - class types
      - interface types
      - array types
      - enum

Variables, Literals and Operators

- Types and Variables
  - What's a variable ?
    - storage location with an associated name
  - Declaration of a variable :
    - 2 components :
      - variable's type
      - his name
  - 2 types of variables :
    - primitive holds the value of exact type
    - reference holds an object or a null reference
  - Variable's value is changed by an assignment

Variables, Literals and Operators

- Primitive types, Values and Literals

Type	Nb bits	Range	Description
Integers			
byte	8	-128 to 127	Byte-length integer
short	16	-32,768 to 32,767	Short integer
int	32	-2,147,483,648 to 2,147,483,647	Integer
long	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,808	Long Integer
Real Numbers			
float	32	0x7f7fffff	Single precision floating-point
double	64	0x7fefffffffffffL	Double precision floating-point
Other Types			
boolean	1	----	A boolean value (true - false)
char	16	----	A single Unicode character

Variables, Literals and Operators

You *really* don't want to spill that...

Be sure the value can fit into the variable.


```
int x = 24;
byte b = x;
//won't work!!
```

Variables, Literals and Operators

- Types and Variables

```
public class Demo
{
   public static void aMethod()
   {
     int aFirstInt = 3;
     int aSecondInt= 5;
     aFirstInt=aSecondInt+8;
     char aChar='d';
     boolean aBool = true;
     String aString="hello";
     Object obj;
   }
}
```

  - Variable name :
    - legal identifier
      - Unicode character beginning with a letter
    - not true, false or null
    - unique in scope

Reserved words

This table reserved.

boolean	byte	char	double	float	int	long	short	public	private
protected	abstract	final	native	static	strictfp	synchronized	transient	volatile	if
else	do	while	switch	case	default	for	break	continue	assert
class	extends	implements	import	instanceof	interface	new	package	super	this
catch	finally	try	throw	throws	return	void	const	goto	enum

Reference types

The 3 steps of object declaration, creation and assignment

```
1          3  2
Dog myDog = new Dog();
```

1 Declare a reference variable

```
Dog myDog = new Dog();
```

Tells the JVM to allocate space for a reference variable, and names that variable *myDog*. The reference variable is, forever, of type Dog. In other words, a remote control that has buttons to control a Dog, but not a Cat or a Button or a Socket.

2 Create an object

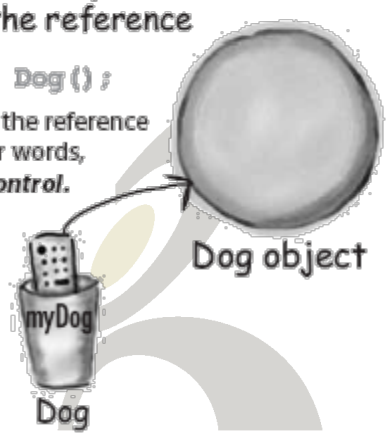
```
Dog myDog = new Dog();
```

Tells the JVM to allocate space for a new Dog object on the heap

3 Link the object and the reference

```
Dog myDog = new Dog();
```

Assigns the new Dog to the reference variable myDog. In other words, ***programs the remote control.***



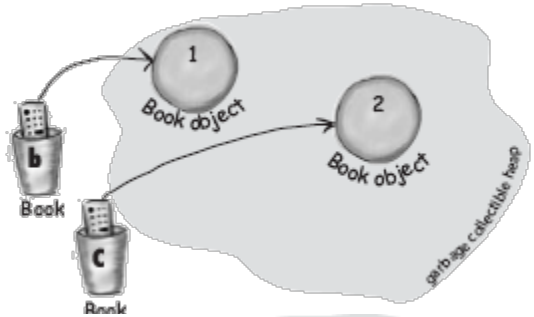
Reference types

```
Dog d = new Dog();
d.bark();
```

think of this like this

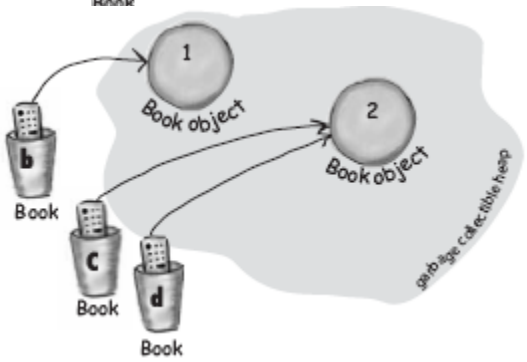
Think of a Dog reference variable as a Dog remote control. You use it to get the object to do something (invoke methods).

Reference types



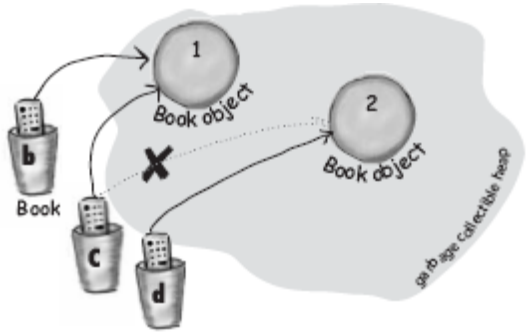

```
Book b = new Book();
Book c = new Book();
```

References : 2
Objects : 2




```
Book d =c;
```

References : 3
Objects : 2




```
c = b;
```

References : 3
Objects : 2

Reference types

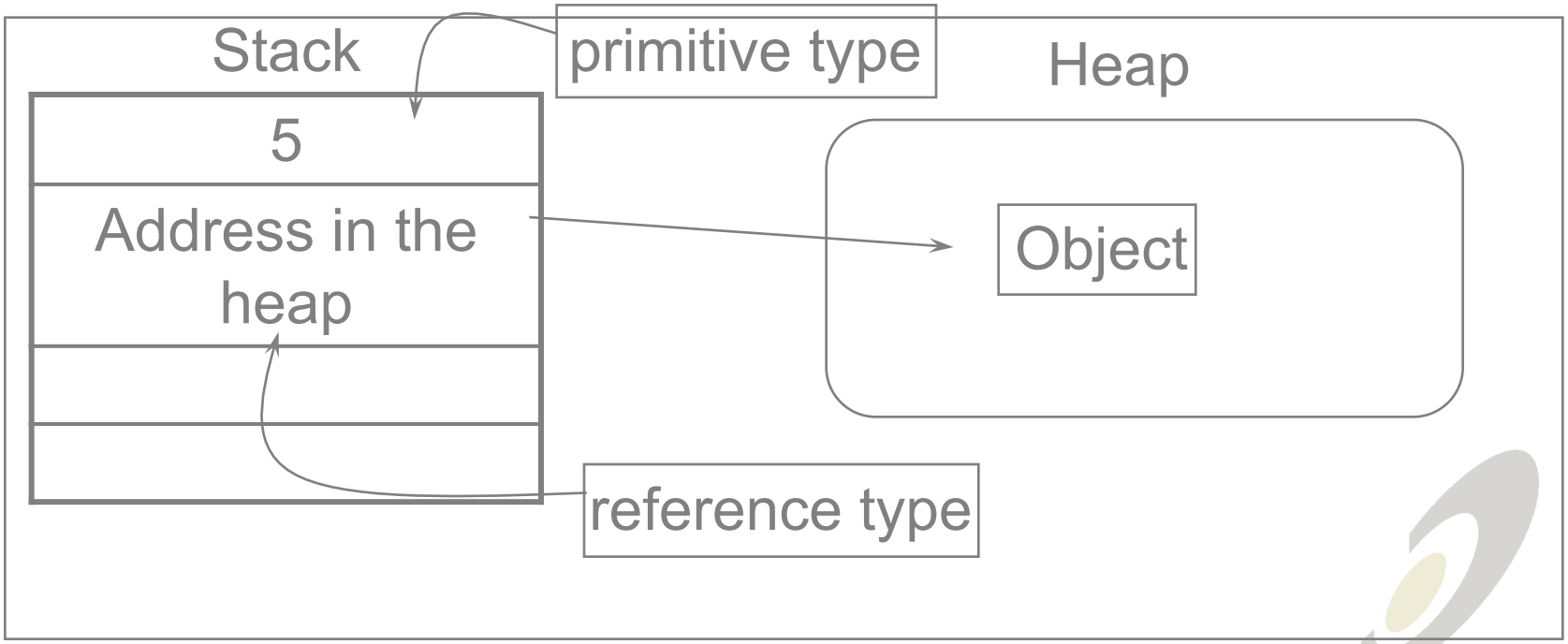
- Lay out of memory
  - The registers
    - Fasted storage inside processor but quantity
    - Not controlled by developer
  - The stacks
    - in RAM with direct support from processor via *stack pointer*
    - Fast and efficient
    - Java compiler knows (when it creates program) exact size and lifetime of all data stored on the stack :
      - limit flexibility of programs
      - Java object references in stack not objects themselves.

Reference Types

- Lay out of memory
  - The heap
    - also in RAM.
    - Java compiler doesn't need to know exact size and lifetime of data stored in the heap.
    - new ➔ storage allocated on the heap
    - Flexibility ⇗ but time to create ⇗
  - Non RAM storage
    - Streamed objects
    - Persistent objects

Reference Types

- The reference type



Reference Types

- Differentiation between reference and primitive types

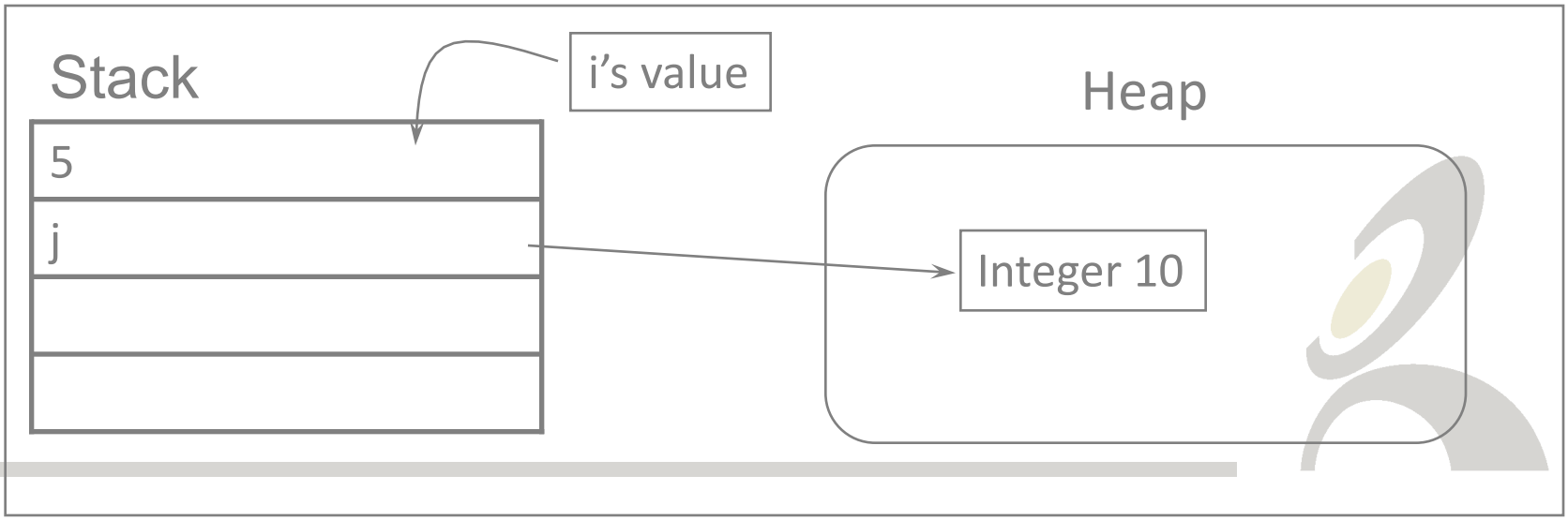
Primitive type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

The Reference Type

- Differentiation between reference and primitive types

```
int i = 5;
Integer j = new Integer(10);
```

- In memory



Reference Types

- Differentiation between reference and primitive types
  - Call a method

```
int i = 5;
Integer j = new Integer(10);
…
i.hashCode()        //Error
j.hashCode()        //OK
```

The String Class

- In Java, Strings are objects

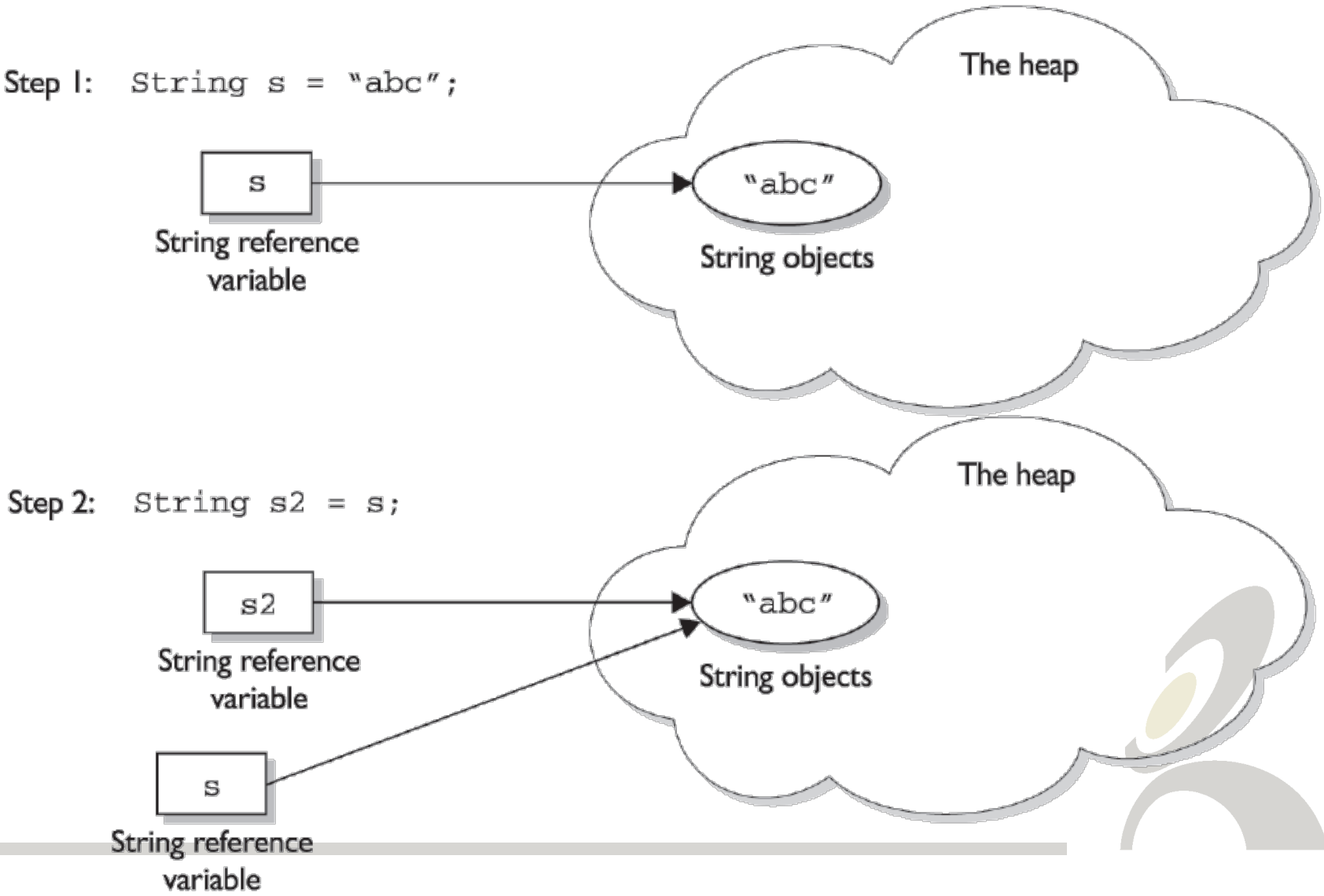
```
String s = new String();

s = "abcdef";

String s = new String("abcdef");

String s = "abcdef";
```

- So far so good…but… Strings are immutable !

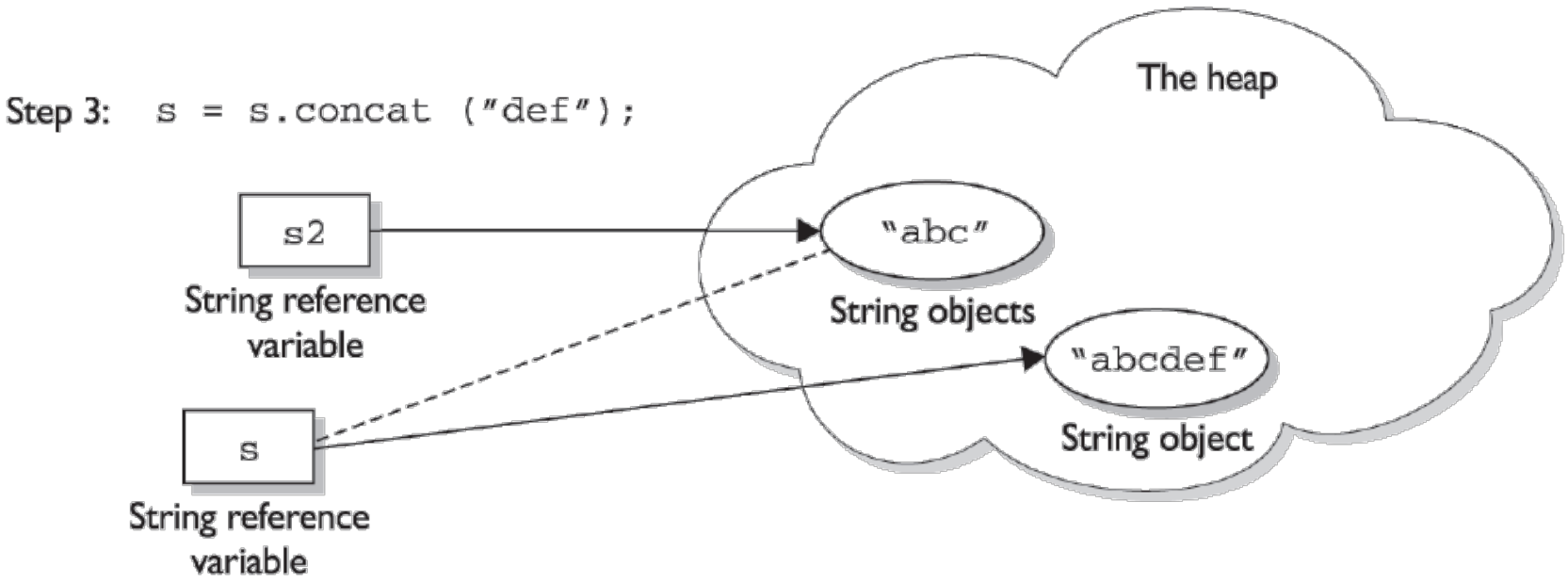
The String Class

- What is immutable?
  - Once a value has being assigned to a String, that value can never change – it's immutable

The String Class



The String Class



The String Class

- String constant pool
- Creating new Strings

```
String s = "abc";
// creates one String object and one reference variable

String s = new String("abc");
//creates two objects and one reference variable
```

The String Class

- Important methods in the String class

charAt()	**Returns the character located at the specified index**
concat()	Appends one String to the end of another ("+" also works)
equalsIgnoreCase()	Determines the equality of two Strings, ignoring case
length()	Returns the number of characters in a String
replace()	Replaces occurrences of a character with a new character
substring()	Returns a part of a String
toLowerCase()	Returns a String with uppercase characters converted
toString()	Returns the value of a String
toUpperCase()	Returns a String with lowercase characters converted
trim()	Removes whitespace from the ends of a String

Use of a StringBuilder

- Lot of manipulations with String objects

  → Abandoned String objects in the String pool

StringBuilder

Use of a StringBuilder

- StringBuilder can be modified over and over again…

```
StringBuilder sb = new StringBuilder("abc");
sb.append("def");
System.out.println(sb);          //abcdef
sb.append("ghi").reverse().insert(3, "---");
System.out.println(sb);          //ihg---fedcba
```

Variables, Literals and Operators

- Primitive types, Values and Literals
  - What's a literal ?
    - source code representation of a value
    - Each type has literals.
      - boolean type has 2 literals : true, false
      - Object reference has 1 literal : null;
  - Integer Literals (3)
    - decimal;
    - octal;
    - hexadecimal;
    - binary

Variables, Literals and Operators

- Primitive types, Values and Literals
  - Integer Literals (3)
    - beginning by 0x or 0X : hexadecimal
      - 0xcafec0ca
    - beginning by 0 : octal
      - 057621
    - beginning without 0 : decimal
      - 35941
    - finish by l or L : long
      - 0xcafec0cal, 076545L, 954581L
    - beginning with 0b : binary
      - 0b00110001
    - an `int` literal assigned to a `short` or `byte` variable, literal is treated like a `short` or a `byte` within valid range

Variables, Literals and Operators

- Primitive types, Values and Literals
  - Floating point literals
    - decimal digit with optionally decimal point followed by E or e and exponent
      - 3.14159, 21.21e21
    - finish with f or F or D
  - Character literals
    - Unicode between single quote
      - 'a'

Literal	Meaning
\n	line feed
\b	backspace
\t	tab
\f	form feed
\r	carriage return
\"	double quote
\'	single quote
\\	backslash

Variables, Literals and Operators

- Primitive types, Values and Literals
  - String literals : a special case
    - zero or more characters enclosed in double quote
      - String str = "peter";
    - not primitive type but instances of `String` class.

Variables, Literals and Operators

- Operators
  - Intro
    - +, -, *, /, =
    - can change value of an operands
      - a = a + b;
    - only with primitives : except ‘=’, ‘==’, ‘!=’
    - String support ‘+’ and ‘+=’
  - Precedence
    - * & / before + & -  ()
      - A = X + Y - 2/2 + Z;

        ≠
      - A = X + (Y - 2)/(2 + Z);

Variables, Literals and Operators

- Operators
  - Assignment =
    - lValue = rValue;
    - Example :

```
public class Demo {

 public static void aMethod()
 {
  int aFirstInt = 3;
  int aSecondInt = 5;
  aFirstInt = aSecondInt + 8;
  char aChar = 'd';
  boolean aBool = true;
  String aString = "hello";
 }

}
```

Variables, Literals and Operators

- Operators
  - Mathematical operators
    - +, -, *, /, % (Remainder of a integer division)
    - Integer division truncates
    - Shorthand notation
      - x += 4;

Variables, Literals and Operators

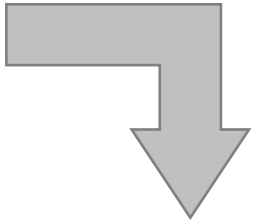
- Operators
  - Mathematical operators
    - Unary minus and plus operators
      - x = -a;
      - x = a * -b;
      - x = a * (-b);

Variables, Literals and Operators

- Operators
  - Mathematical operators
    - Auto increment and decrement
      - a++;
      - a--;
      - ++a;
      - --a;

Variables, Literals and Operators

```
public class AutoInc {
 public void perform() {
  int i = 1;
  prt("i : " + i);
  prt("++i : " + ++i); // Pre-increment
  prt("i++ : " + i++); // Post-increment
  prt("i : " + i);
  prt("--i : " + --i); // Pre-decrement
  prt("i-- : " + i--); // Post-decrement
  prt("i : " + i);
 }
 private void prt(String s) {
  System.out.println(s);
 }

 public static void main(String[] args) {
  AutoInc ai = new AutoInc();
  ai.perform();
 }
}
```



Output

```
i : 1
++i : 2
i++ : 2
i : 3
--i : 2
i-- : 2
i : 1
```

Variables, Literals and Operators

- Operators
  - Mathematical operators
    - Relational operators
      - >
      - <
      - <=
      - >=
      - ==
      - !=

Variables, Literals and Operators

- Operators
  - Mathematical operators
    - Relational operators
      - >
      - <
      - <=
      - >=
      - ==
      - !=

Variables, Literals and Operators

- Operators
  - Mathematical operators
    - Relational operators
      - &&, ||, !
      - Short-circuiting

Variables, Literals and Operators

- Operators
  - String concatenation (+)

```
String string1 = "box " + (5 + 3);// Result:"box 8"

String string2 = ("box " + 5) + 3;// Result:"box 53"

String string3 = "box " + 5 + 3;// Result:"box 53"

String string4 = 5 + " box " + 3;// Result:"5 box 3"
```

Variables, Literals and Operators

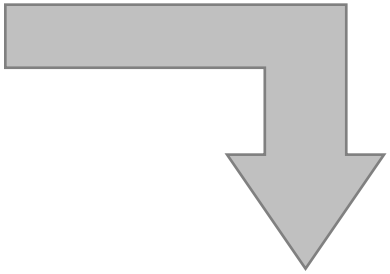
- Operators
  - Casting operators
    - Why cast ?
  - Promotion
    - When you use 2 different types for a Mathematical or bitwise operation, the result type is the same of the bigger type

```
public class Casting {

 public void cast() {
   int i = 200;
   long l = (long) i;
   long l2 = (long) 200;
 }

}
```

Variables, Literals and Operators

- Operators
  - The problem of overflow

```
public class Overflow {
  public static void main(String[] args) {
    int big = 0x7fffffff; // max int value
    prt("big = " + big);
    int bigger = big * 4;
    prt("bigger = " + bigger);
  }

  static void prt(String s) {
    System.out.println(s);
  }
}
```



  - Java is good, but it's not that good.

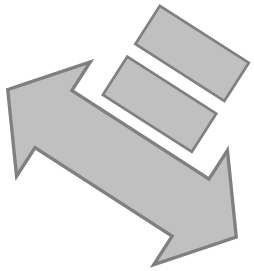
Variables, Literals and Operators

- Operators
  - Bits Manipulation
    - &, |, ^, ~ or &=, |=, ^= but ~=
    - <<, >>, >>> or <<=, >>=, >>>=

Variables, Literals and Operators

- Operators
  - Ternary if-else operator
    - boolean-exp ? value0 : value1

```
static int ternary(int i) {
 return i < 10 ? i * 100 : i * 10;
}
```



```
static int alternative(int i)
{
  if (i < 10)
    return i * 100;
  else
    return i * 10;
}
```

Variables, Literals and Operators

- Operators
  - Operator Precedence and Associativity

Variables, Literals and Operators

Symbol	Note	Precedence (highest number = highest precedence)
++ --	pre-increment, decrement	16
++ --	post-increment, decrement	15
~	flip the bits of an integer	14
!	logical not (reverse a boolean)	14
- +	arithmetic negation, plus	14
(type name)	type conversion (cast)	13
* / %	multiplicative operators	12
- +	additivie operators	11
<< >> >>>	left and right bitwise shift	10
instanceof < <= > >=	relational operators	9
== !=	equality operators	8
&	bitwise and	7
^	bitwise exclusive or	6
|	bitwise inclusive or	5
&&	conditional and	4
||	conditional or	3
? :	conditional operator	2
= *= /= %= -= += <<= >>= >>>= &= ^= |=	assignment operators	1

Variables, Literals and Operators

- Operators
  - Complex Expressions
    - Use parentheses if clarify
    - maximize your code's readability

    Or

    - use intermediate variable with descriptive name

Variables, Literals and Operators

- Operators
  - Order of Evaluation
    - Expression evaluate Left to Right
    - Every operand to an operator evaluated before operator

    Except
    - &&, ||, ?:.
      - Right side not evaluate if left side determines the result

Questions ??