Lecture 4: Keyboard Input, Mathematical Function, Operators, String

Sub: Object Oriented Programming

Code: CSE-1205

Taking input from the keyboard

- Here Scanner class is used to take input from the keyboard.
- Scanner is a simple text scanner which can parse primitive types and strings using regular expressions.
- First, Scanner class is connected to System.in
- Then, it uses it's internal functions to read from System.in
- Scanner class is under the package of java.lang.util
- Example:

```
Scanner sc = new Scanner(System.in);
int i;
If(sc.hasNextInt()==true)
   i = sc.nextInt();
 else{
 }
```

Take an input from the keyboard

```
import java.util.*;
public static void main(String[] args) {
 double value;
   System.out.print(“Enter a floating point
 number:");
 Scanner stdin = new Scanner(System.in);
 if(stdin.hasNextDouble()==true)
    value=stdin.nextDouble();
 System.out.println(“You have entered:
 ”+value);
}
```

Scanner API

```
public Scanner(InputStream in)      // Scanner(): convenience constructor for an
                                    // InputStream

public Scanner(File s)              // Scanner(): convenience constructor for a filename

public int nextInt()                // nextInt(): next input value as an int

public short nextShort()            // nextShort(): next input value as a short

public long nextLong()              // nextLong(): next input value as a long

public double nextDouble()          // nextDouble(): next next input value as a double

public float nextFloat()            // nextFloat(): next next input value as a float

public String next()                // next(): get next whitespace-free string

public String nextLine()            // nextLine(): return contents of input line buffer

public boolean hasNext()            // hasNext(): is there a value to next
```

Another Example

```
import java.util.*;

public class MathFun {

  public static void main(String[] args) {
       // set up the Scanner object
       Scanner stdin = new Scanner(System.in);

       // have the user input the values for x and y
       System.out.print("Enter a decimal number: ");
       double x = stdin.nextDouble();
       System.out.print("Enter another decimal number: ");
       double y = stdin.nextDouble();

       double squareRootX = Math.sqrt(x);

       System.out.println ("Square root of " + x + " is "
                           + squareRootX);
  }
}
```

Java Operators

- 4 groups:
  - Arithmetic
  - Bitwise
  - Relational
  - logical

Arithmetic Operators

- Operators: +, -, *, /, %, ++, --, +=, -=, *=, /=, %=
- For integers:
  - **Division is integer division**
    - 6 / 2 yields 3
    - 7 / 2 yields 3, not 3.5
  - **Modulus** is %
    - Returns the remainder
    - 7 % 2 yields 1
    - 6 % 2 yields 0
- For Floats and doubles
  - **Division**
    - 7.0 / 2.0 yields 3.5
    - 7.0 / 2 yields 3.5
    - 7 / 2.0 yields 3.5
    - 7 / 2 yields 3

- **Modulus**:
  - Differs from C/C++
  - 47.5 % 10 yields 7.5

+=, ++

- += is more efficient than +
  - A=A+5; A+=5;
- ++
  - Increments a number variable by 1
- --
  - Decrements a numeric variable by 1
- Output:

```
int i = 4,j,k;
j=++i; //prefix form
k=i++; //postfix form
```

Bitwise operators

- Bitwise operator
  - | - or
  - & - and
  - ~ - Not
  - ^ - XOR
  - >> - shift right
  - << - shift left
  - >>> - shift right zero fill

Example of Bitwise and Relational Operator

- byte a=8, b=24, c ;
- c=a | b;
- c=a & b;
- c=~a;
- c=a^b;
- c=a<<1;
- c=a>>2;
- c=a>>>1;

Results

00001000 | 00011000 = 24

00001000 & 00011000 = 8

~00001000=11110111=-120

00001000 ^ 00011000 = 16

00001000 << 1= 16

00001000 >> 2= 2

00001000 >>> 1= 4

Relational Operator

- Relational operators:==, !=, >, <, >=, <=
- The outcome of these operations is a boolean value. So the outcome is not numeric value.
- true and false are non-numeric values.
- Example 1:

```
int a=4, b=1;
boolean c= a<b;                    //The result of c will be false
```

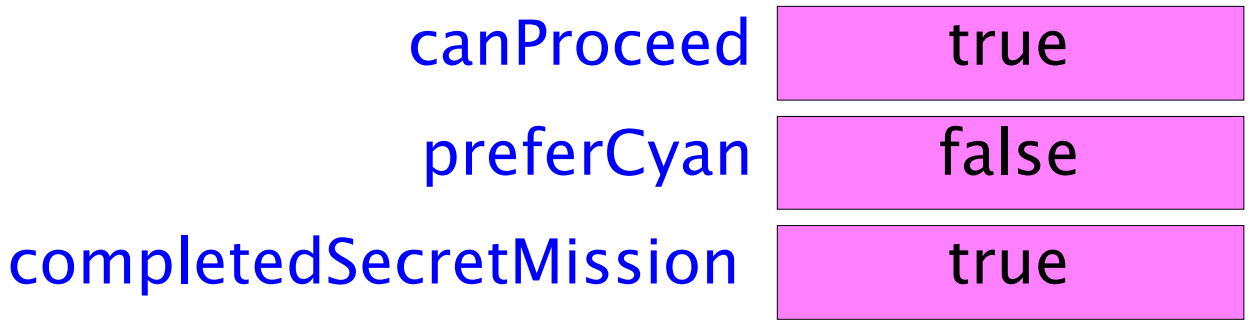
- Example 2:

```
int done = 3;
if(done)                                    //error
        System.out.println("abc");
  if(done!=0)                               //ok
        System.out.println("abc");
```

Defining boolean variables

- Local boolean variables with initialization

```
boolean canProceed = true;
boolean preferCyan = false;
boolean completedSecretMission = true;
```



Boolean Logical Operators

- Works on boolean values.
- Operators: |, &, ^, ||, &&, !, ?:, !=, ==,
- Suppose

```
boolean p = true;
boolean q = false;
boolean r = true;
boolean s = false;
```

- What is the value of

```
p                    p && s
!s                   p == q
q                    q != r
p && r               r == s
q || s               q != s
```

Evaluating boolean expressions

- Suppose

```
int i = 1;
int j = 2;
int k = 2;
char c = '#';
char d = '%';
char e = '#';
```

- What is the value of

j == k	i != k
i == j	j != k
c == e	d != e
c == d	c != e

Short-circuit Logical Operators

- || and && are the short-circuit operators.
  - Java will not evaluate the right-hand operand when the outcome of the expression can be determined by the left operand alone.
- | and & are not the short-circuit operators.
  - Java evaluates both sides of the operator.
- Example:
  - if(a!=0 && b/a > 10) {...}
    - If a==0, right part will not be evaluated.
  - if(a!=0 & b/a > 10) {...}
    - If a==0, the program returns run-time exception.
- We should use || and &&

Assignment vs. Comparison

- = is the assignment operator
- == is the comparison operator
  - Returns a boolean (true or false) if the two sides are equal
  - Consider:

```
int x = 5;
System.out.println (x == 5);
System.out.println (x == 6);
```

  - Prints out true, false

Operator precedence revisited

- Highest to lowest
  - Parentheses
  - Unary operators
  - Multiplicative operators
  - Additive operators
  - Relational ordering
  - Relational equality
  - Logical and
  - Logical or
  - Assignment

Mathematical Functions

- The **Math** class contains methods for common math functions.
- They are *static* methods, meaning you can invoke them using the "Math" class name (more on "static" later).

```
// compute the square root of x
double x = 50.0;
double y = Math.sqrt( x );
```

This means: the sqrt( ) *method in the* Math *class.*

```
// raise x to the 5th power ( = x*x*x*x*x)
double x = 50.0;
double y = Math.pow( x , 5 );
```

Mathematical Functions

Common Math Functions

abs(x)	absolute value of x
cos(x), sin(x), tan(x)	cosine, sine, etc. x is in *radians*
acos(y), asin(y), atan(y), ...	inverse cosine, sine, etc.
toDegrees(radian)	convert radians to degrees
toRadians(degree)	convert degrees to radians
ceil(x)	ceiling: round *up to nearest int*
floor(x)	floor: round *down to nearest int*
round(x)	round to the nearest integer
exp(x)	exponential: $y = e^x$
log(y)	natural logarithm of y ($y = e^x$)
pow(a, b)	a^b (a to the power b)
max(a, b)	max of a and b
min(a, b)	min of a and b

Examples of Using Math Functions

Expression	Result	Type of result
`Math.sqrt( 25.0 );`	5.0	double
`Math.sqrt( 25 );`	5.0	double
`Math.log( 100 );`	4.60517018	double
`Math.log10( 100.0 );`	2.0	double
`Math.sin( Math.PI/2 );`	1.0	double
`Math.cos( Math.PI/4 );`	0.70710678	double
`Math.abs( -2.5 );`	2.5	double
`Math.abs( 12 );`	12	int
`Math.max( 8, -14);`	8	int
`Math.min( 8L, -14L);`	-14L	long
`Math.max( 8.0F, 15);`	15F	float
`Math.pow( 2, 10 );`	1024.0	double
`Math.toRadians( 90 );`	1.5707963	double
`Math.E;`	2.7182818...	double
`Math.PI;`	3.1415926...	double

Overloaded Math Functions

- Some methods in Math have multiple implementations for different parameter types.

`abs(x)`	returns "int" if x is "int"; returns "long" if x is long; returns "float" if x is float; returns "double" if x is "double".
`max(a,b)`	returns "int" if a *and* b are "int"; returns "long" if a *and* b are "long"; etc.
`round(x)`	returns "float" if x is float; returns "double" if x is "double".
`but...`	
`sqrt(x)`	*always* promotes x to double and returns a double. Most math functions are like this (sin, cos, tan, log, log10, ...).

overload*:* using the same name for functions that have different parameters.

Example: Math.abs(int) has int parameter and returns an int result.

Math.abs(double) has double parameter and returns a double

Overloaded Functions Example

Example	Returns
`Math.max( 2, 10 )`	`(int) 10`
`Math.max( -1L, -4L )`	`(long) -1L`
`Math.max( 2F, 10.0F )`	`(float) 10.0F`
`Math.max( -4.0, 0.5 )`	`(double) 0.5`

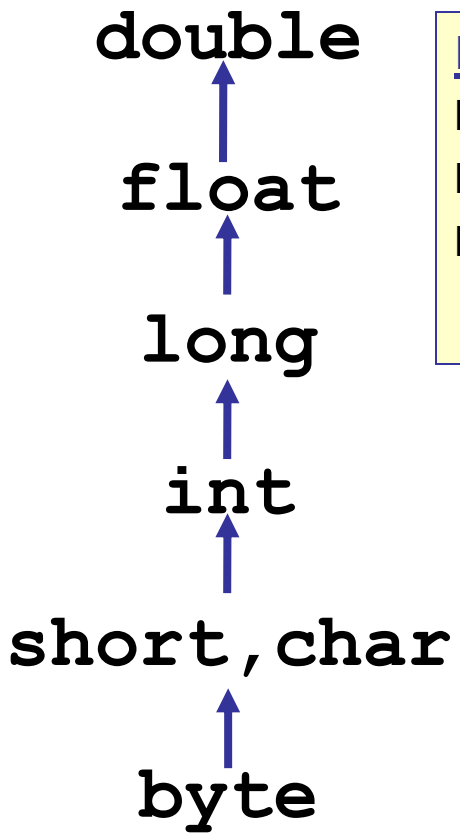
What if the arguments are of different date types?

What should be the data type of the returned value?

Example	Returns
`Math.max( 2, 10.0F )`	?
`Math.max(-1, -4L )`	?
`Math.max( 3, 1.25 )`	?

Functions and Data Types

Java *promotes* one of the arguments until it finds a matching function *prototype*.



Example	Promotion	Then Call
`Math.max( 2, 10.0F )`	`2 to 2.0F`	`max(2F, 10F)`
`Math.max(-1, -4L )`	`-1 to -1L`	`max(-1L, -4L)`
`Math.max( 3, 2.236 )`	`3 to 3.0`	`max(3.0,2.236)`

Automatic Conversions

When necessary, Java automatically "promotes" an argument to a higher data type according to the diagram. These widening conversions will never "overflow" the data type, but may result in lose of precision

Analyzing an Expression

How would you write this in Java syntax?

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

Hint: use Math.sqrt(desc)

Your answer:

_____

Analyzing an Expression

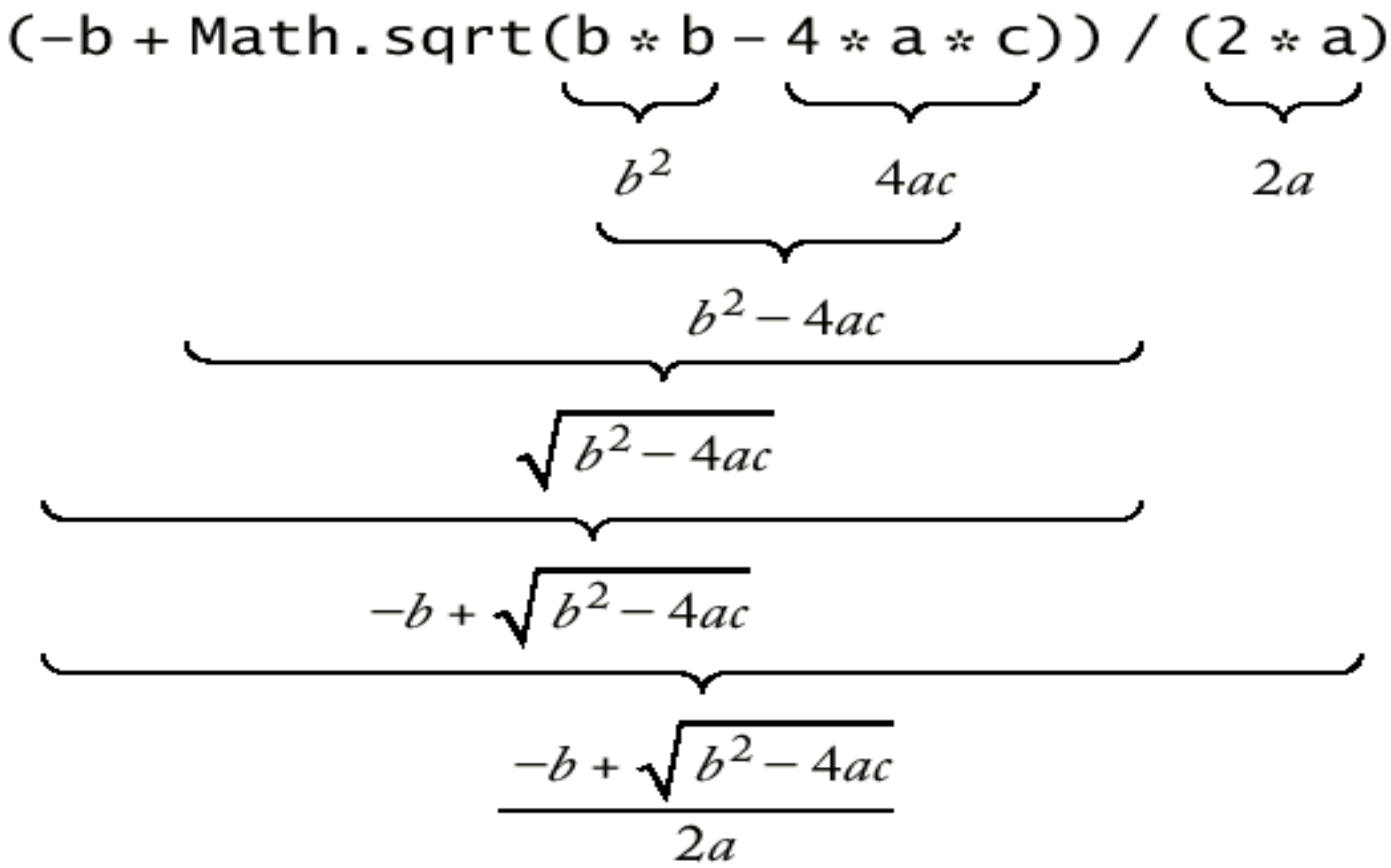
How would you write this in Java syntax?

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

In what order would Java evaluate this expression:

```
x = ( -b + Math.sqrt(b*b - 4*a*c) ) / ( 2 * a)
```

Analyzing an Expression



Strings

- Java provides a class definition for a type called String
- Since the String class is part of the java.lang package, no special imports are required to use it (like a header file in C).
- Just like regular datatypes (and like C), variables of type String are declared as:
  - String s1;
  - String s2, s3; //etc.
- Note that String is uppercase. This is the Java convention for classnames.

Strings

- Initializing a String is painless
  - s1 = “This is some java String”;
- Note that double quotes are required.
- Think of above method as shortcut for more standard way (assuming s1 has been declared):

  - s1 = new String(“This is some java String”);
  - *new* operator required to create memory for new String object.

String

- Two ways to create a string
- As a new object through the

    - String class
        - String name = new String("text");

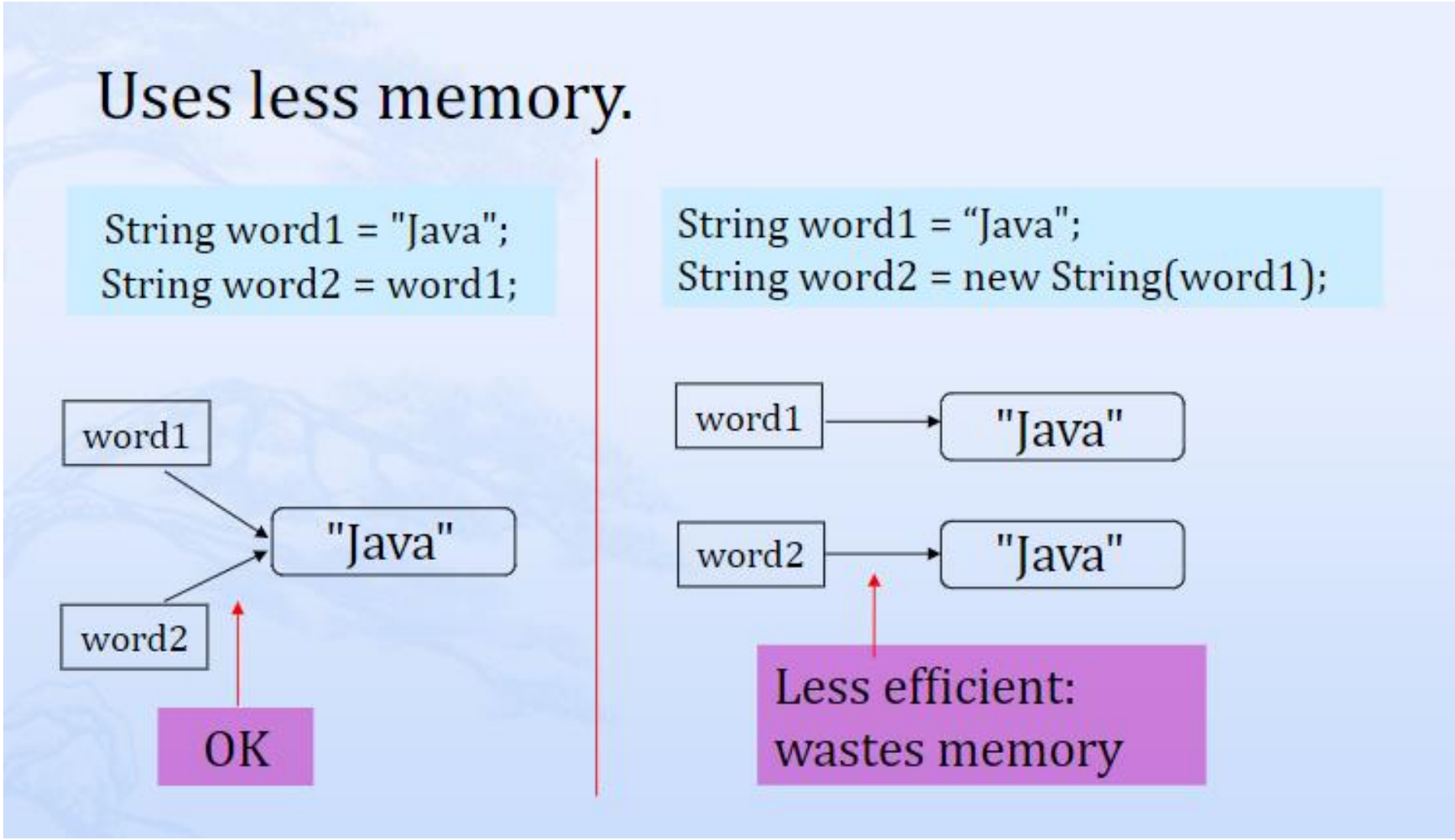
    - Convenient shorthand method
        - String name = "text";

String Immutability

- String name = "text";
- Once created in memory, a string cannot be changed: none of its methods changes the string
- If a string variable is reassigned, the memory address is deleted and a new memory address is created

- Immutable objects are convenient because several references can point to the same object safely: there is no danger of changing an object through one reference without the others being aware of the change.

String Immutability Advantages



String Index

- Characters of a string are numbered with 0-based *indexes*

```
String name = "Java fun";
```

index	0	1	2	3	4	5	6	7
character	J	a	v	a		f	u	n

  - First character's index : 0
  - Last character's index : 1 less than the string's length

  - The individual characters are values of type `char`

String Function

Method name	Description
indexOf(str)	index where the start of the given string appears in this string (-1 if not found)
length()	number of characters in this string
substring(index1, index2) or substring(index1)	the characters in this string from index1 (inclusive) to index2 (exclusive); if index2 is omitted, grabs till end of string
toLowerCase()	a new string with all lowercase letters
toUpperCase()	a new string with all uppercase letters
charAt(index)	Returns the character at the specified index

String Length

- It is sometimes useful to determine the length of a string
- Use the length( ) methods

  - String s = "Java is Fun";
  - int sLength = s.length();
  - System.out.println(sLength); // 11

- The method counts the number of characters in the string variable. It produces an integer result.

Modifying String

- These methods build and return a new string, rather than modifying the current string.
- To modify a variable's value, you must reassign it:

    - String s = "Hello World";
    - s1 = s.toUpperCase();
    - s2 = s.toLowerCase();
    - System.out.println(s1 + " " + s2);// HELLO WORLD

- hello world

Looking for a String

- String name = "President George Washington";

- name.indexOf ('P');
- name.indexOf ('e');
- name.indexOf ("George");
- name.indexOf ('e', 3);
- name.indexOf ("Bob");
- name.lastIndexOf ('e');

Replacing String

- We may also want to find and replace parts of a large string
- There are a few replace methods available
  - replace(oldString, newString)
  - replaceFirst(oldString, newString)
  - replaceAll(oldString, newString)

- String s = "Java is fan";
- s1 = s.replace("fan","fun");
- System.out.println(s1);// Java is fun

Compare String

- **equals() method**
  - Evaluates contents of two String objects to determine if they are equivalent
  - Returns true if objects have identical contents
- **equalsIgnoreCase() method**
  - Ignores case when determining if two Strings equivalent
- s1 = "Java";
- s2 = "java";
- boolean eq = s1.equals(s2);
- System.out.println(eq); // false

Homework (String and Scanner)

- Write a program that takes two string S1 and S2 as input and perform the following operations:
  1. Print the length of each string.
  2. Replace all spaces of S1 to underscore(_).
  3. Print the first character of S1.
  4. Compare the string S1 and S2 and print “equal” or “not equal” accordingly
  5. Find the first occurrence of character ‘a’ in S1 and print it’s position.
  6. If S1 is a substring of S2 or S2 is a substring of S1 then print a message.
  7. Convert the S1 string to lower case and S2 string to upper case letter.
  8. Save the S1 string to a character array.

1. What is the task of “trim” function?