



Getting Started with Java

CS 18000

Sunil Prabhakar

Department of Computer Science

Purdue University

[Objectives]

In this module we will:

- Learn some basic Java
- Explore some standard classes
 - String
 - JOptionPane
 - Scanner
 - System
 - GregorianCalendar, Calendar

[A Basic Java Program]

- Every Java program is implemented as a collection of Class definitions.
- The simplest program is simply one class.
- We will write a program to simply say “Hello World” on the screen.
- Our program will consist of the HelloWorld class.
 - defined in a file called HelloWorld.java

[Program HelloWorld]

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

1. Copy the above text into a file called *HelloWorld.java*
2. Compile the program using the command:
 %javac HelloWorld.java
3. Run the program using the command:
 %java HelloWorld

[An Interactive HelloWorld]

```
import java.util.Scanner;

public class HelloWorld {
    public static void main(String[] args) {
        Scanner myScanner;
        String name;

        myScanner = new Scanner(System.in);

        System.out.println("Enter your name");

        name = myScanner.next();
        System.out.println("Hello " + name);

    }
}
```

[Java Programs]

- Each class is defined in a file with extension .java
- A program is made up of statements that each end with a ; (or sometimes }
- Statements are made up of
 - words e.g., **public class new** myScanner
 - and special symbols e.g., { } () ; , < > & ! |
- Some words have a special meaning in Java. These are called **reserved words**. Shown in **bold blue** in the slides.

[Java Program Structure]

- In Java, a program is a collection of classes.
- Each class is defined in a .java file
 - Sometimes, more than one class is defined in a single file
 - We may use some existing classes by importing them into our program

[Parts of A Simple Program]

An optional import

Optional comments

The definition of a class

```
import java.util.Scanner;

/**
 * This Class demonstrates a simple Java Program
 * Created on 7/7/16
 */

public class HelloWorld {
    public static void main(String[] args) {
        Scanner myScanner;
        String name;

        myScanner = new Scanner(System.in);

        System.out.println("Enter your name");

        name = myScanner.next();
        System.out.println("Hello " + name);

    }
}
```


[Comments]

- Comments can appear anywhere in the program
- A comment starts with `/*` and ends with the next `*/`
 - Everything from the start to the end of the comment is ignored by the compiler
 - They are only for helping us understand the program better
 - Note: no nesting of comments allowed
- Single line comments can start with `//` and end at the end of the line.

Examples of comments

```
/*  
    All of this is ignored  
    import java.util.Scanner;  
  
    no matter what or how many lines, until we get  
    to the end of comment marker */ import  
java.util.Scanner;
```

```
// a single line comment  
import java.util.Scanner;  
// Another single line comment
```

```
/*  
    * This class demonstrates a  
    * simple Java program.  
    * Created by sunil on 7/8/16.  
    */
```

```
/*  
    All of this is ignored  
    import java.util.Scanner;  
    /* no nested comments allowed */  
    no matter what or how many lines, until  
    we get to the end of comment marker */  
  
import java.util.Scanner;
```



Comment ends here

[Examples of comments]

- Comments are important for readability, and can help with documenting your program
- Will often be omitted from the slides due to space limitations
- will be present in the example programs
- A special class of comments called javadoc comments are extremely useful and can automatically generate documents (e.g., java API)

```
/**  
 * This class demonstrates a  
 * simple Java program.  
 * Created by sunil on 7/8/16.  
 */
```

Start with /**

[Parts of A Simple Program]

An optional import

Optional comments

The definition of a class

```
import java.util.Scanner;

/**
 * This Class demonstrates a simple Java Program
 * Created on 7/7/16
 */

public class HelloWorld {
    public static void main(String[] args) {
        Scanner myScanner;
        String name;

        myScanner = new Scanner(System.in);

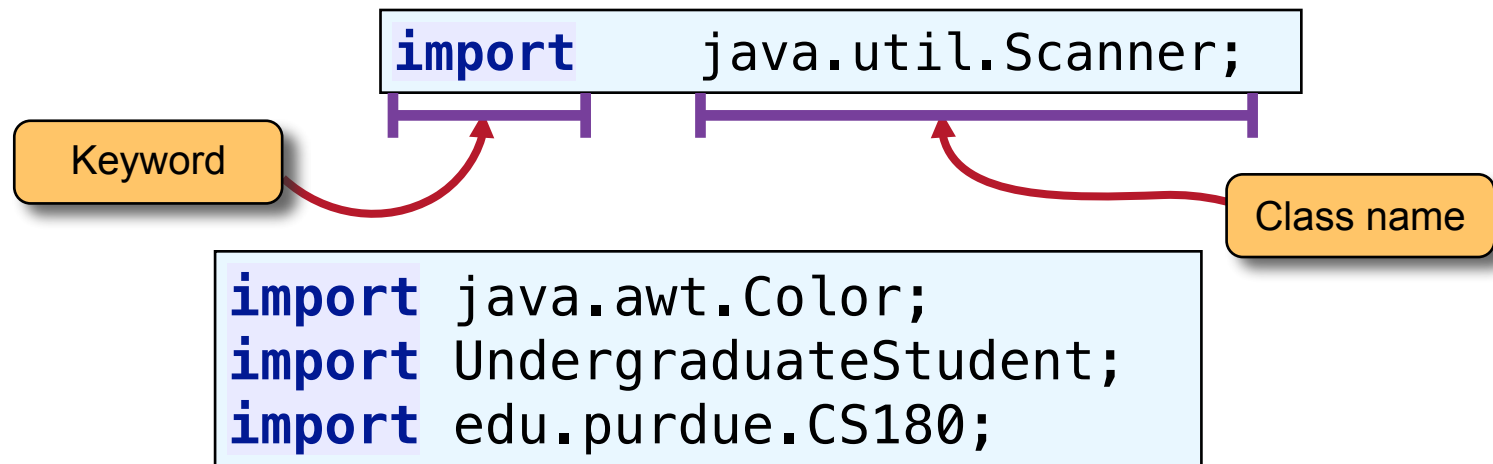
        System.out.println("Enter your name");

        name = myScanner.next();
        System.out.println("Hello " + name);

    }
}
```

[Import Statement]

In order to use most existing classes, we have to explicitly import them into our program.

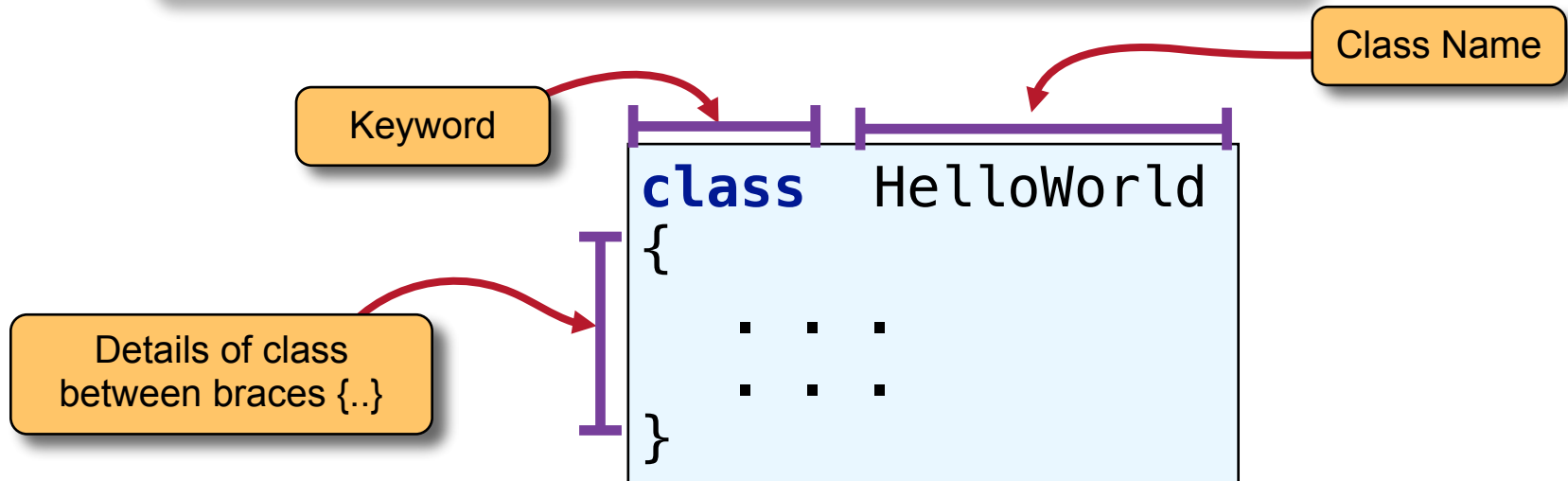


Classes have special naming to show where they are located (more later).

******Some classes do not need to be imported (e.g., `System`, `String`)

[Class Definition]

A class is declared using the **class** keyword and the name of the class



By convention, class names always start with a capital letter.

Typically, a class is stored in a file with its name, e.g., HelloWorld.java

[The main() Method]

- Each java program must declare at least one special method: the main method.
- The program starts execution at the first line of this method.
- Each line of code in this method is executed in order.
- After the last line is executed, the program ends.

[The main() Method]

The main() method

```
public class HelloWorld {  
    public static void main(String[] args) {  
        Scanner myScanner;  
        String name;  
  
        myScanner = new Scanner(System.in);  
        System.out.println("Enter your name");  
        name = myScanner.next();  
        System.out.println("Welcome " + name);  
    }  
}
```


[Main Method Definition]

Keywords

```
public static void main(String[] args) {  
    . . .  
    . . .  
}
```

The body of the method is defined between the braces.

The main method must be declared as follows. Later in the course we will understand what each piece.

[Manipulating Data]

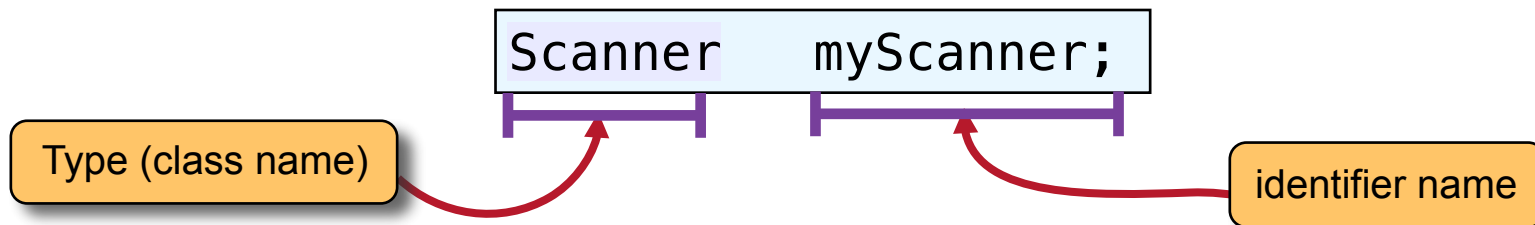
- Most programs manipulate data in various ways.
- In order to do this, we need to be able to access data items in our code.
- We use identifiers to refer to data items.
- An **identifier** refers to a specific piece of memory. The content of this piece of memory may change during program execution, hence identifiers are also called **variables**.
- Java is **Strongly Typed**: every identifier must have a specific type that is explicitly declared to the compiler
 - Results in more reliable code

[Identifiers]

- An identifier
 - Cannot be a reserved word
 - Can consist only of letters(A..Z,a..z), digits(0..9), \$ and _
 - Cannot begin with a digit
 - Best to avoid \$ and _
- These are required rules. We also have Java naming conventions that make programs easier to read
 - Identifiers begin with a lowercase letter
 - Class names begin with an uppercase letter
 - We use “camel case” for multiple words
- Good identifier names (typically nouns) give a hint about the role of the data.

[Declaring an Object Identifier]

In order to use an object, we need a way to refer to it. We explicitly declare the name and type of the identifier in our program.



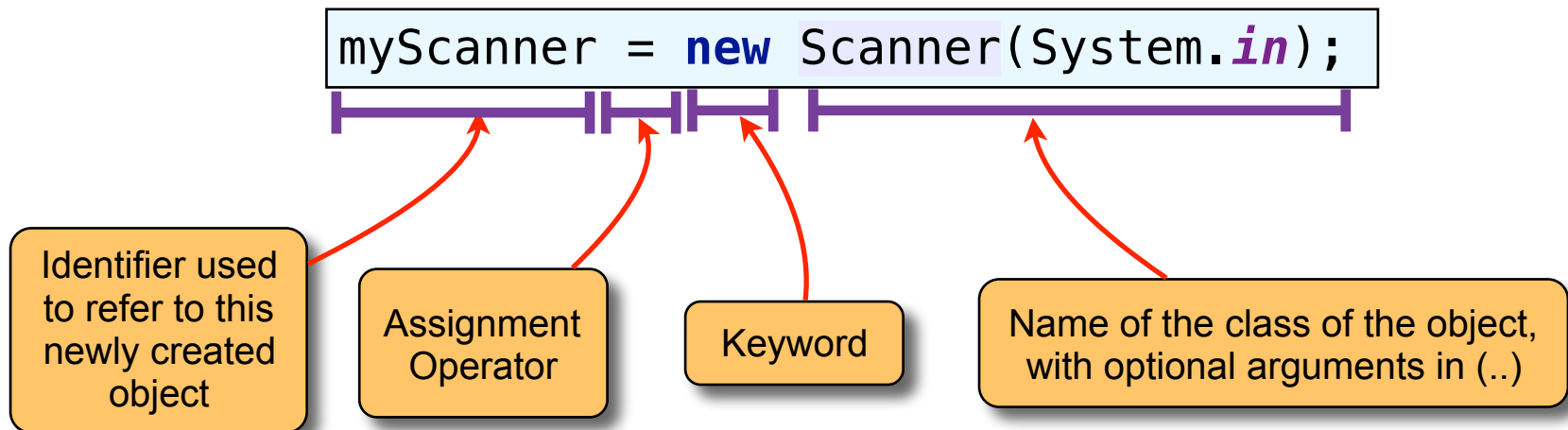
By convention, identifier names begin with lower case letters.

Multiple identifiers of the same type can be declared together.

```
JFrame myWindow;  
String userName;  
Student sally, bob, newStudent;
```

[Creating an object]

Before we can use an object in our program, we must create the object. We use the keyword **new** to create an object.



[Assignment]

Important to know what is happening in memory!

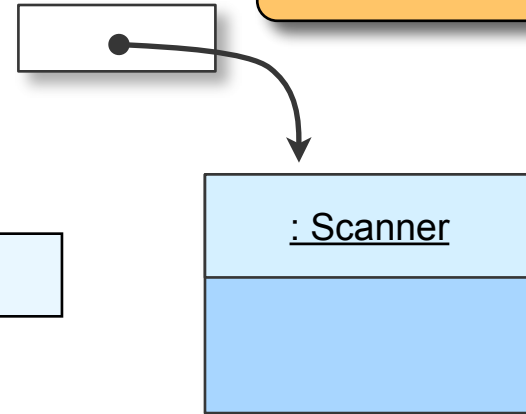
```
Scanner myScanner;
```

```
myScanner = new Scanner(System.in);
```

The code to the right of “=” is computed first and the result is copied (assigned) to the identifier to the left of “=”

For the new operator, the result is the location in memory of the newly created object.

myScanner



The identifier is declared and associated with a location in memory

[Manipulating Objects]

- Once we have an object created, we can perform actions on the object.
- Each of these actions corresponds to a method defined in the class to which the object belongs.
- In order to execute a method, we use a reference to an object.

[Calling a Method]

- Each method has a name and may take one or more arguments.
- To execute a method, we use its name followed by parentheses ()
- Any arguments to the method are place between these parentheses
 - Multiple arguments are separated by commas

[Using an object]

We can use an object by calling methods on it.

```
System.out.println("Hello ");
```

Reference
to the
object

Note the . between
the object and the
method name.

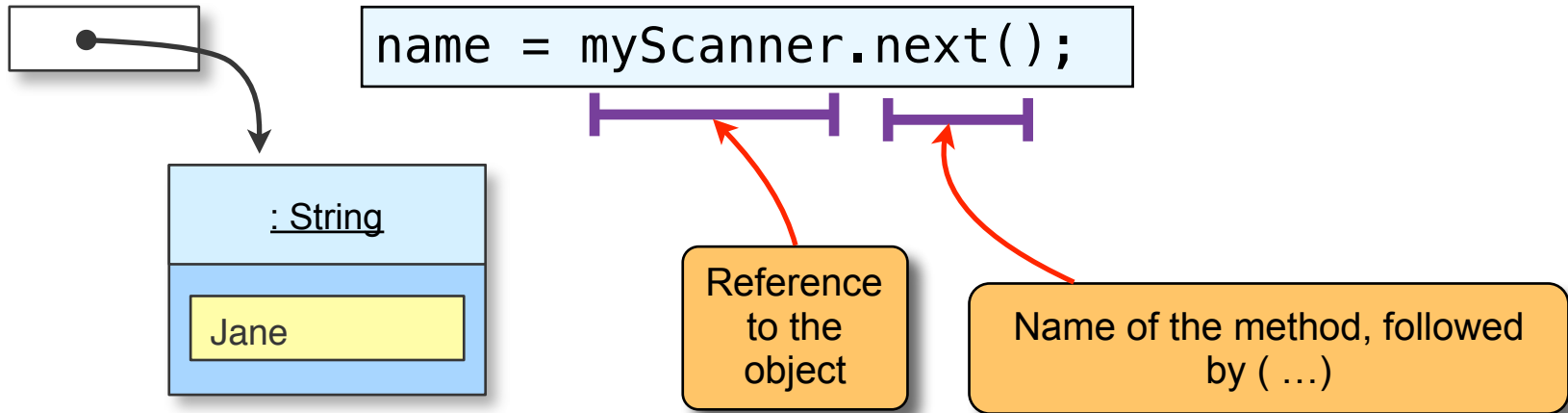
Name of the method,
followed by parentheses,
and sometimes arguments
between the parentheses.

We are calling the println method on the System.out object and passing it the argument "Hello".
This should cause the program to print out Hello to the screen.

Note: The System.out object is automatically available to our program without explicit declaration or creation.

[Another Example]

name



The result of calling this method is copied to (assigned) the `name` reference. In this example, an object of the class `String` is returned, and its location is saved in the reference called `name`.

[Note]

- Java is case-sensitive
 - window and Window are different
- The color and font of the text are meaningless
 - their purpose is to make programs easier to read for humans
- Words are separated by whitespace (spaces, line breaks, tabs)
 - multiple consecutive whitespaces are ignored by the compiler
 - only useful for readability of programs

[Identical Programs (to Compiler)]

```
import java.util.Scanner;

/**
 * This class demonstrates a simple Java program.
 * Created by sunil on 7/8/16.
 */
public class HelloWorld {
    public static void main(String[] args) {
        Scanner scanner;
        String name;

        scanner = new Scanner(System.in);

        System.out.println("Enter your name");

        name = scanner.next();
        System.out.println("Hello " + name);

    }
}
```

```
import java.util.Scanner; public class HelloWorld { public static void
main(String[] args) { Scanner scanner; String name; scanner = new
Scanner(System.in); System.out.println("Enter your name"); name = scanner.next();
System.out.println("Hello " + name);}}
```

[Important]

Not Declared!

Will not compile!

- All identifiers used in a program must be declared before being used.

```
public class HelloWorld {  
    public static void main(String[] args) {  
        Scanner myScanner;  
        String name;  
  
        myscanner = new Scanner(System.in);  
  
        System.out.println("Enter your name");  
  
        name = myscanner.next();  
        System.out.println("Hello " + name);  
  
    }  
}
```

[Important]

- If an object identifier is not initialized to a correct object, you cannot call any methods on it.
- You can only reference an object of the same class as the declared class of the identifier.

Not Initialized!

```
public class HelloWorld {  
    public static void main(String[] args) {  
        Scanner myScanner;  
        String name;  
  
        myScanner.next();  
  
        myScanner = new String("Hello");  
        System.out.println("Enter your name");  
  
        name = myScanner.next();  
        System.out.println("Hello " + name);  
    }  
}
```

Wrong Class!

[Common Java Practice]

Use a different name to avoid confusion.

- It is very common to use the name of the class without capitalization as the name of a variable of this class.
- Remember: Java is case sensitive.

Note!

```
import java.util.Scanner;

/**
 * This class demonstrates a simple Java program.
 * Created by sunil on 7/8/16.
 */
public class HelloWorld {
    public static void main(String[] args) {
        Scanner scanner;
        String name;

        scanner = new Scanner(System.in);

        System.out.println("Enter your name");

        name = scanner.next();
        System.out.println("Hello " + name);

    }
}
```

Be careful

[Caution: Smart Editors]

- A string is defined using double quotes:
"abcd"
 - Many “smart” text editors will automatically change these to fancier characters: “abcd”
 - The compiler will not recognize these fancy quotes and will throw errors -- be careful.
 - Fix these quotes if you cut and paste any code.

[Using Standard Classes]

- Java comes with a rich set of classes already defined to serve many common needs
- We can easily create useful programs using these classes
- The Java API describes standard classes (linked from the web page)
- We will motivate the use of some of these classes to solve some simple problems

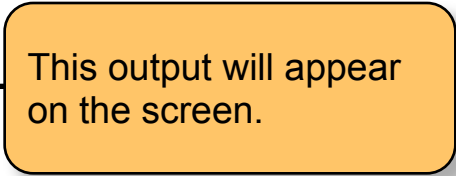
[Standard Output]

- Using the **print** method of the **System.out** object is a simple way to write to the console window from which the program was run.

```
System.out.println("Hello World!");
```



>Hello World!



This output will appear on the screen.

[Multiple Lines]

We can display multiple lines of text by separating lines with a new line marker `\n`, or by using the **println** method.

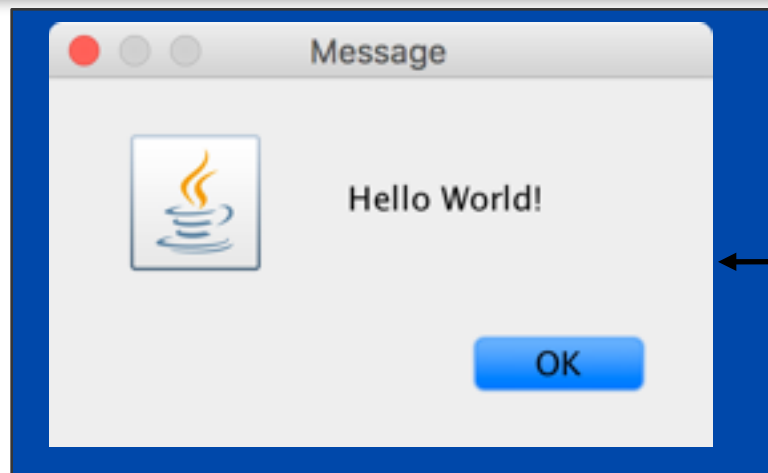
```
System.out.print("Course Number: ");  
System.out.println("CS18000");  
System.out.print("Title: ");  
System.out.println("Introduction to Problem Solving\n and OOP");
```

```
Course Number: CS18000  
Title: Introduction to Problem Solving  
and OOP
```

[JOptionPane]

Using **showMessageDialog** of the **JOptionPane** class is a simple way to bring up a window with a message.

```
JOptionPane.showMessageDialog(null,  
                             "Hello World!");
```

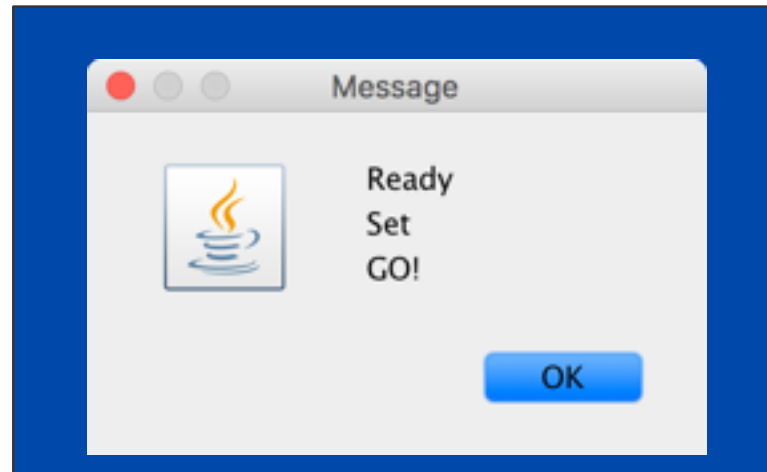


This dialog will appear at the center of the screen.

[Displaying Multiple Lines of Text]

We can display multiple lines of text by separating lines with a new line marker `\n`.

```
JOptionPane.showMessageDialog(null, "Ready\nSet\nGO!");
```



[Standard Input and Scanner]

- The **System** class has a special object that accepts input from the keyboard: **System.in**
- It reads only one byte at a time. We often need to read multiple bytes at a time.
- The **Scanner** class provides the necessary methods.
- A scanner object is created that “wraps” the **System.in** object.
- Calls to the method **next()** return one “word” at a time from the standard input
- Words are separated by whitespaces.

[Standard Input and Scanner]

```
import java.util.Scanner;

public class HelloWorld {
    public static void main(String[] args) {
        Scanner myScanner;
        String name;

        myScanner = new Scanner(System.in);

        System.out.println("Enter your name");

        name = myScanner.next();
        System.out.println("Hello " + name);

    }
}
```

```
> Enter your name Lisa ↵
> Hello Lisa.
```

Reading in multiple words

```
import java.util.*;
...

Scanner scanner;
String firstName, lastName;
scanner = new Scanner(System.in);
System.out.print("Enter your first and last name: ");
firstName = scanner.next();
lastName = scanner.next();
System.out.println("Hello " + lastName + ", " + firstName + ".");
```

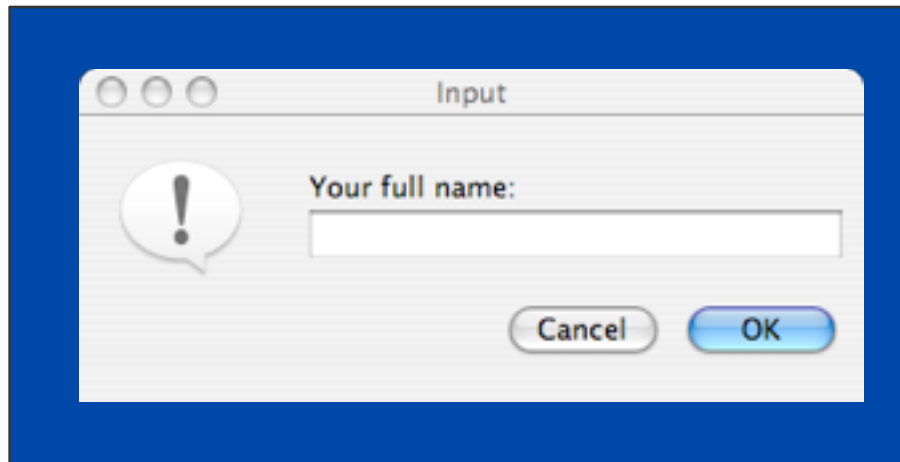
Enter your first and last name: Yinian Qi
Hello Qi, Yinian.

[JOptionPane for Input]

Using **showInputDialog** of the **JOptionPane** class is another way to input a string.

```
String name;
```

```
name = JOptionPane.showInputDialog(null, "Your full  
name:");
```



This dialog will appear at the center of the screen ready to accept an input.

[The *String* Class]

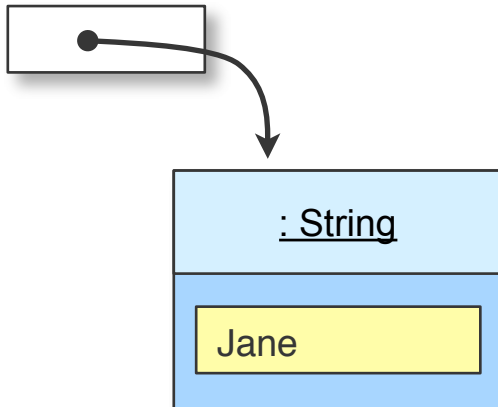
- The textual values passed to the showMessageDialog method are instances of the String class.
- A sequence (string) of characters separated by double quotes is a String constant or String literal.
- There are close to 50 methods defined in the String class. We will introduce three of them here: substring, length, and indexOf.
- We will also introduce a string operation called **concatenation: +**

[Creating a String Object]



```
String str;  
str = new String ("Jane");
```

str



Memory is allocated and the identifier str is associated with it.

A String object with the value "Jane" is created, and str is set to reference this new object.

[Concatenating Strings]

- We have seen how the + operator can be used to combine two string objects into a new string that is the concatenation of the two strings.
- The two strings being combined remain unchanged.
- Either one can be a string literal or object.
- A new string is created.

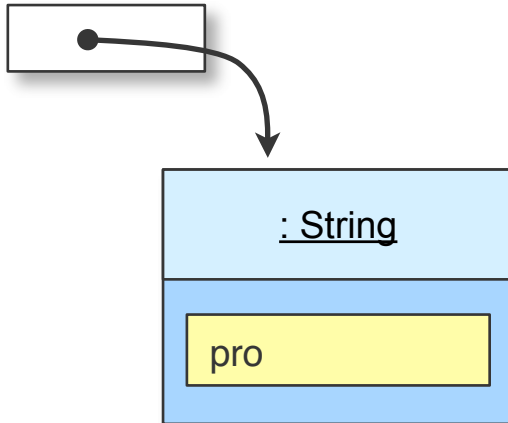
[String Concatenation]



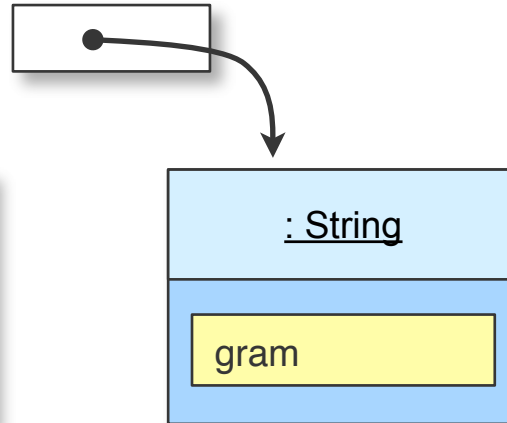
```
String str1, str2, str3;  
str1 = new String ("pro");  
str2 = "gram";  
str3 = str1 + str2;
```

Only for String: we do not need to call **new** to create a String object!

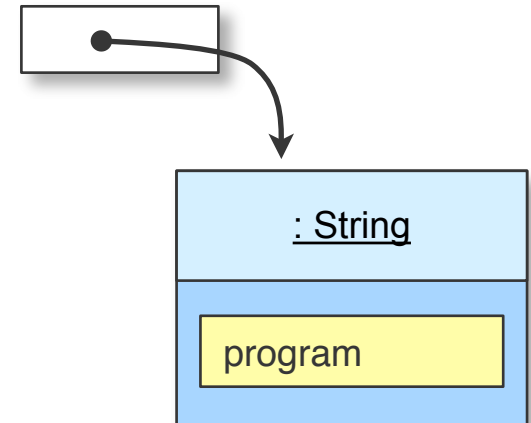
str1



str2



str3



[Examples: concatenation]

```
String str1, str2;  
str1 = new String ("Hello");  
str2 = "World";
```

str1 + str2

→ HelloWorld

str1 + " " + str2

→ Hello World

str2 + ", " + str1

→ World, Hello

str2 + "! " + str1 + "!"

→ World! Hello!

[The substring() Method]

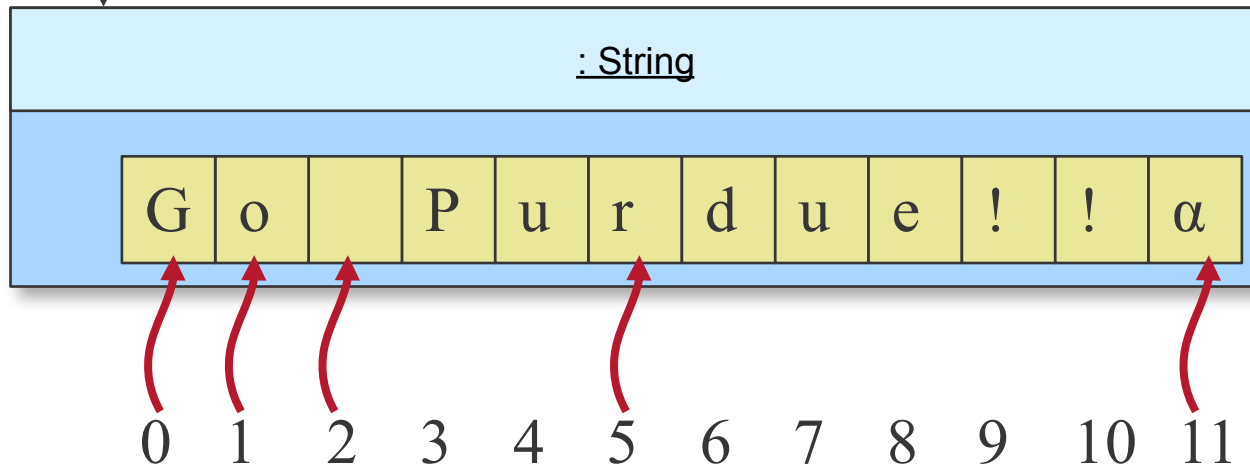
- It is possible to extract a part of a string using the substring() method
- Remember that a string is a sequence of unicode characters
- Each character can be referenced using its position, or **index**, in this sequence
- Indexing begins with 0
- The string reference must be correctly initialized, or else the program will crash

[Character Indexing]

text

```
String text;  
text = "Go Purdue!!α";
```

Only for String: we do not need to call **new** to create a String object!



Each character has an index, starting with 0

[Examples: substring()]

```
String text = "Purdue!!";
```

`text.substring(6,8)` → `"!!"`

`text.substring(0,8)` → `"Purdue!!"`

`text.substring(1,5)` → `"urdu"`

`text.substring(3,3)` → `""`

`text.substring(4,2)` → `error`

[The substring() Method]

- This method is called on an existing string (usually, a string reference)
- It usually takes two arguments which are numbers
 - the first is the starting index from which the substring should begin
 - the second is one more than the ending index at which the substring should end
- the second must not be larger than the first
- Both must be between 0 and the number of characters in the string
- A new string containing a copy of the selected characters is the result
 - The original string remains unchanged.

[The length() Method]

- Given a string reference, initialized to a string object, we can determine the length of the string currently referenced using the `length()` method.
- This method is called on the string reference.
- If the reference is not initialized, our program will crash.

[Examples: length()]

```
String str1, str2, str3,  
str4;  
str1 = "Purdue!!";  
str2 = "Purdue Pete" ;  
str3 = "" ; //empty string  
str4 = " " ; //one space
```

str1.length()	→	8
str2.length()	→	11
str3.length()	→	0
str4.length()	→	1

[Parsing a Simple Token String]

- A comma separated values or CSV format is often used to save structured data as a string.
 - E.g., to export an excel worksheet
- Also, in reverse, we sometimes need to take a CSV string and extract out the different fields.
- To do this we need to pull out substrings without knowing the exact index values to use
 - they depend upon the length of the values in the input.
- To achieve this separation, we can use the `indexOf ()` method of the String class

[CSV Example]

- Consider a CSV string with fields corresponding to first, middle, and last names, and birthday information
 - Alan,Mathison,Turing,23,June,1912
 - Grace,Brewster,Hopper,9,December,1906
- Each field, is separated by a comma
- `indexOf ()` gives us the first index in the string on which it is called, at which the string passed as an argument occurs.
 - If it doesn't occur even once, it returns -1
 - the search is case-sensitive

[CSVParser Step 1]

```
public class SimpleCSVParser {  
    public static void main(String[] args) {  
        Scanner scanner;  
        String unparsedInput;  
        String firstName;  
  
        scanner = new Scanner(System.in);  
        System.out.println("Enter the Comma Separated Values String:");  
  
        unparsedInput = scanner.next();  
  
        System.out.println("The first comma occurs at index:" +  
                           unparsedInput.indexOf(","));  
    }  
}
```

[CSVParser Step 2]

```
public class SimpleCSVParser {
    public static void main(String[] args) {
        Scanner scanner;
        String unparsedInput;
        String firstName;

        scanner = new Scanner(System.in);
        System.out.println("Enter the Comma Separated Values String:");

        unparsedInput = scanner.next();

        firstName = unparsedInput.substring(0, unparsedInput.indexOf(","));

        System.out.println("First name:" + firstName);

        unparsedInput = unparsedInput.substring(unparsedInput.indexOf(",")+1,
                                                unparsedInput.length());

        System.out.println("unparsed input is now:" + unparsedInput);
    }
}
```


[CSVParser Step 3]

```
...
firstName = unparsedInput.substring(0, unparsedInput.indexOf(","));
unparsedInput = unparsedInput.substring(unparsedInput.indexOf(",")+1,
                                         unparsedInput.length());

middleName = unparsedInput.substring(0, unparsedInput.indexOf(","));
unparsedInput = unparsedInput.substring(unparsedInput.indexOf(",")+1,
                                         unparsedInput.length());
...

dobMonth = unparsedInput.substring(0, unparsedInput.indexOf(","));

dobYear = unparsedInput.substring(unparsedInput.indexOf(",")+1,
                                  unparsedInput.length());

System.out.println("First name:" + firstName + "\n" +
                  "Middle name: " + middleName + "\n" +
                  "Last name: " + lastName + "\n" +
                  "Day: " + dobDay + "\n" +
                  "Month: " + dobMonth + "\n" +
                  "Year: " + dobYear + "\n");
```

[The Date Class]

- The **Date** class from the **java.util** package is used to represent a date.
- When a **Date** object is created, it is set to the current date set in the computer
- The class has a **toString()** method that converts the internal format to a string.

```
Date now;  
now = new Date();  
System.out.println(now.toString());
```

→ Tue Aug 09 16:39:29 EDT 2016

[Problem Statement]

Create a dialog box that accepts a user's login and password and prints a record indicating the ID and time of login attempt.

This is a very simple problem.

[Overall Plan]

- To solve this problem, we first break the problem down into sub-problems:
 - Get the user's login name
 - Get the user's password
 - Get the date and time
 - Display the ID and time

[Development Steps]

- We will develop this program in three steps:
 1. Start with the program template and add code to get input
 2. Add code to obtain the time
 3. Write the output

[Step 1 Design]

- The program specification states “accepts the user’s ID and password” but doesn’t say how.
- We will consider “how” in the Step 1 design
- We will use JOptionPane for each input.

[Step 1 Get Inputs From User]

```
import javax.swing.*;
/**
 * A simple class to record a login.
 * Created by sunil on 8/9/16.
 */
public class Login {
    public static void main(String[] args) {
        String loginName;
        String password;

        loginName = JOptionPane.showInputDialog(null,
            "Enter your Login ID:");
        password = JOptionPane.showInputDialog(null,
            "Enter your password:");
    }
}
```

[Step 1 Test Inputs]

```
import javax.swing.*;
/**
 * A simple class to record a login.
 * Created by sunil on 8/9/16.
 */
public class LoginStep1 {
    public static void main(String[] args) {
        String loginName;
        String password;

        loginName = JOptionPane.showInputDialog(null,
            "Enter your Login ID:");
        password = JOptionPane.showInputDialog(null,
            "Enter your password:");
        System.out.println("Login:" + loginName +
            " Password:" + password);
    }
}
```


[Step 2 Design]

- We now obtain the current date and time.
- We can get this directly from the computer using the standard Date class.
- Date is defined in the java.util package -- so we have to import that class or package.

[Step 2 Code]

```
import javax.swing.*;
import java.util.*;
/**
 * Created by sunil on 8/9/16.
 */
public class Login {
    public static void main(String[] args) {
        String loginName;
        String password;
        Date now;

        loginName = JOptionPane.showInputDialog(null,
            "Enter your Login ID:");
        password = JOptionPane.showInputDialog(null,
            "Enter your password:");
        now = new Date();
    }
}
```

[Step 3 Display the output]

- We now display the output as required.
- We can do this using `println()`.

[Step 3 Code]

```
import javax.swing.*;
import java.util.*;
public class Login {
    public static void main(String[] args) {
        String loginName;
        String password;
        Date now;

        loginName = JOptionPane.showInputDialog(null,
            "Enter your Login ID:");
        password = JOptionPane.showInputDialog(null,
            "Enter your password:");
        now = new Date();

        System.out.println(loginName + " attempted to login at "
+ now.toString());

    }
}
```

[Problem]

On what day of the week were you born?

- Calendars are complicated: different days in months, leap years, etc.
- The *GregorianCalendar* class can help
- We can create a *GregorianCalendar* object corresponding to any given year, month, and day.
- Note: Months start at 0 for January

GregorianCalendar Example

```
import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.Locale;

/**
 * Created by sunil on 7/8/16.
 */
public class DayOfWeek {

    public static void main(String[] args) {
        GregorianCalendar someDate;
        String dayOfWeek;
        Locale locale = new Locale("EN");

        someDate = new GregorianCalendar(1998, 1, 14); //Note: 1 for February

        dayOfWeek =
            someDate.getDisplayName(Calendar.DAY_OF_WEEK,
                                   Calendar.LONG, locale);

        System.out.println("2/14/1998 was a " + dayOfWeek);
    }
}
```