

# Classes and Objects

Objective:

In this unit, the students will gain an understanding of the difference between a class and an object



X

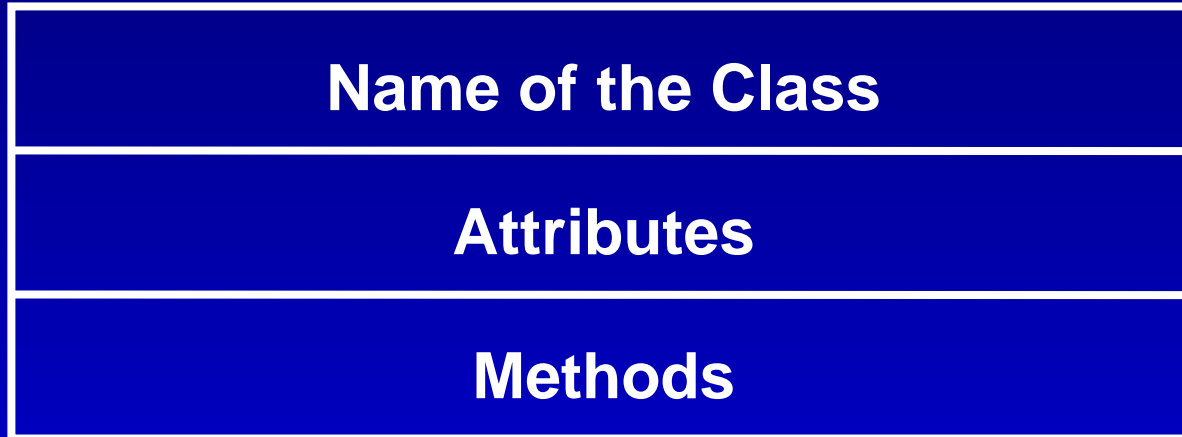
# What is a Class?

- As per Sun Definition
  - A class is a blueprint, or prototype, that defines the variables and the methods common to all objects of a certain kind
- A class is a blueprint for creating objects
- A class defines
  - Attributes
    - Data part
  - Methods
    - A function defined in the class
    - Provides the simple interface
- Once a class has been compiled, objects can be created from that class, and that class becomes a new type



# What is a Class?

- Lets represents by a compartmentalized rectangle



# What is a Class?

## FullTime

List the data for a FullTime employee  
Use the following format -  
variableName: dataType

List the functionality for a FullTime employee  
Use the following format -  
methodName: returnValue

# What is a Class?

- A suggested answer is shown below

<b>FullTime</b>
<b>firstName:String</b> <b>middleName:String</b> <b>lastName:String</b> <b>salary:double</b>
<b>print():String</b> <b>computeGrossPay():double</b>

# Class Declaration Syntax

- [modifiers] **class** *ClassName* [extends *ParentClass*] [implements *Interface1*, *Interface2*...]
- *ClassName*
  - May be any legal identifier
  - Class names, by convention, start with uppercase letters and use capitalization to denote the beginning of the next word (known as camel notation)
- A pair of curly braces { } following the declaration mark the beginning and end of the class declaration
- Inside the curly braces define the
  - Attributes or variables
  - Methods
- Note: Label end curly braces!

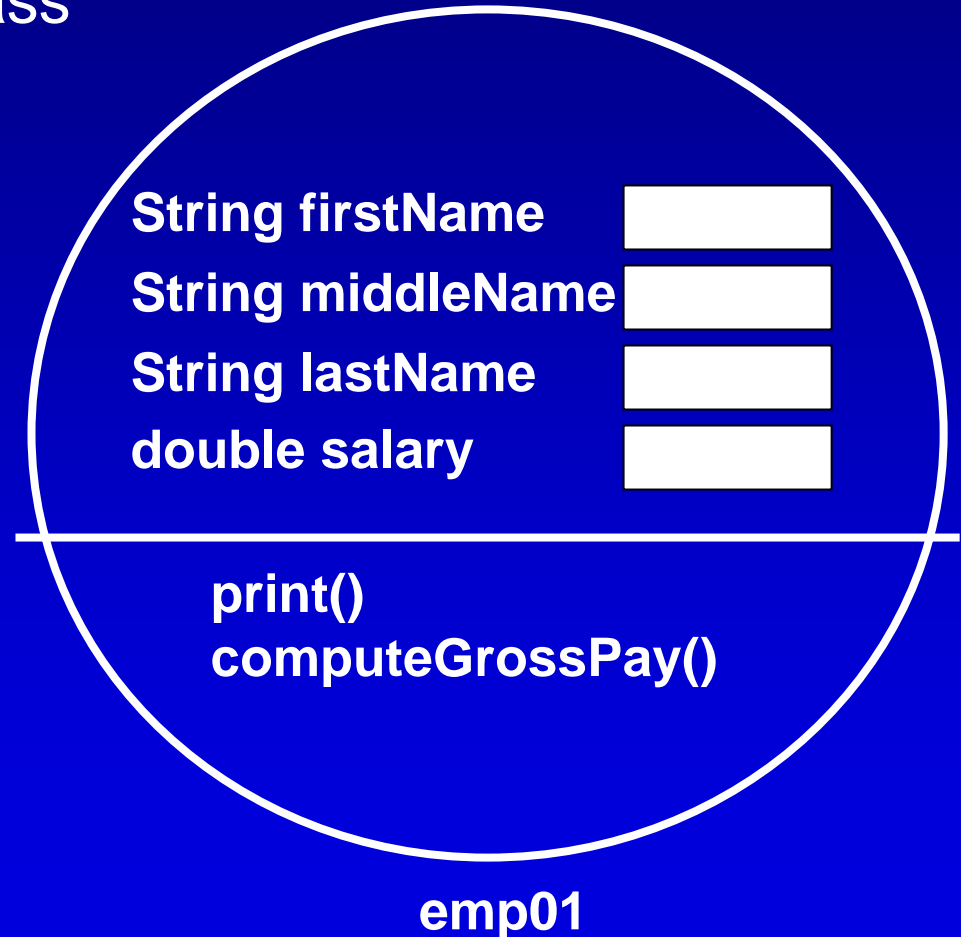
# Class Declaration Syntax

- The code is shown below

```
public class FullTime {  
    String firstName;  
    String middleName; Attributes  
    String lastName;  
    double salary;  
  
    public String print() {  
    } Methods  
  
    public double computeGrossPay() {  
    }  
}
```

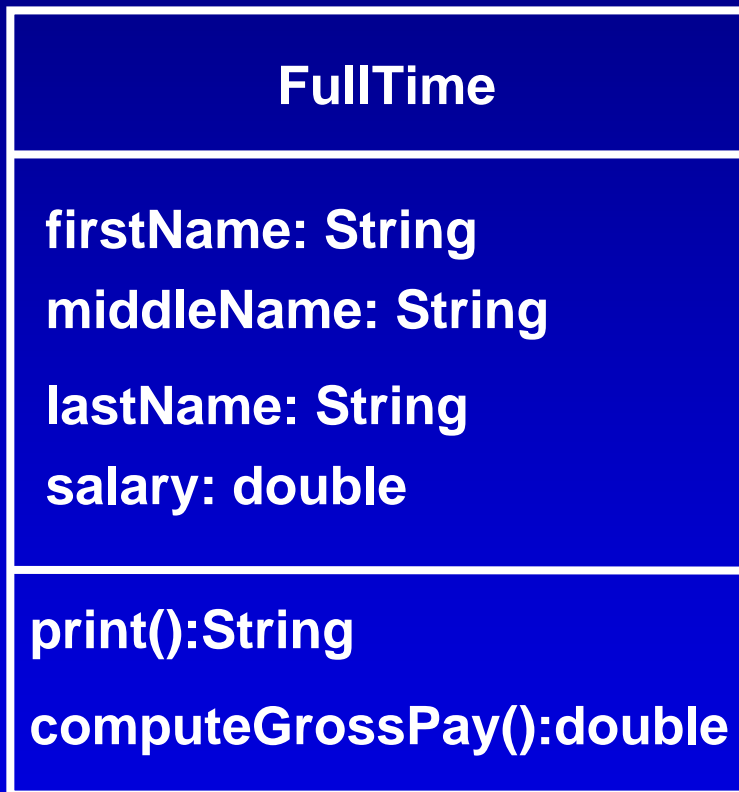
# What is an Object?

- As per Sun's definition
  - An object is a software bundle of variables and related methods based on a class
- An instance of a class

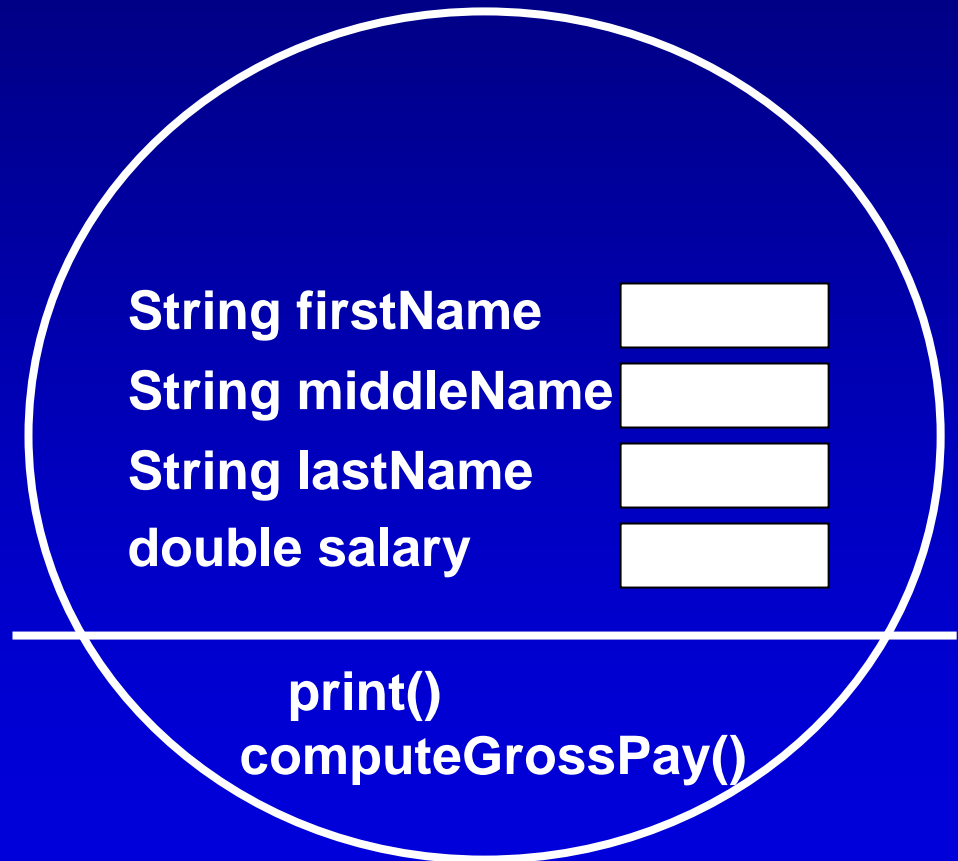


# What is an Object?

- In our given example, if we create an object based on the FullTime class then



the FullTime class



An instance of the FullTime class

# Attributes

- Variables that store data of the class
- Define the state of the class
- Attributes may be known by other names:
  - Data members
  - Attributes
  - Fields
  - Property

# Attribute Definition Syntax

- *[modifiers] dataType variableName;*
  - *modifiers*
    - Access modifiers
      - public – any class may access
      - private – can only be accessed within the same class
      - default – no access modifier (explained later)
      - protected
    - Others
      - static – explained later
  - *dataType*
    - Primitive data types or any class
    - Can be any legal Java identifier

# Attribute Definition Syntax

- An example
  - Declared within the body of the class

```
public class FullTime {  
    private String firstName;  
    public String middleName;  
    String lastName;  
    . . . . .  
}
```

access modifier

data type

attribute name

terminated by a semi-colon

The diagram shows a code block for a class named FullTime. It contains three attribute declarations: private String firstName;, public String middleName;, and String lastName;. The first line of the class body is followed by five dots representing other attributes. A red circle highlights the semi-colon at the end of the first line. Three callout boxes point to the words 'private', 'String', and 'firstName' in the first line, labeled 'access modifier', 'data type', and 'attribute name' respectively. A fourth callout box points to the semi-colon, labeled 'terminated by a semi-colon'.

# Methods

- Are used to break down complex tasks into smaller, more focused processes
- Are functions defined inside a class
- Can have their own local variables
- A method may return:
  - A value
  - A reference to an object
  - Or nothing at all
    - By using the `void` keyword

# Methods

- Methods are similar to constructs in many other languages:
  - Paragraphs in Cobol
  - Subroutines in Basic / VB
  - Procedures or functions in Pascal
  - Functions in C or Fortran
- User-defined methods allow us to add tools that perform some predefined task

# Method Definition Syntax

- *[modifiers] returnType methodName(parameterType parameterName, parameterType parameterName)*
  - *modifiers*
    - Access modifiers
      - public – any class may access
      - private – can only be accessed within the same class
      - default – no access modifier (explained later)
      - protected (explained later)
    - Others
      - final, static, abstract – explained later

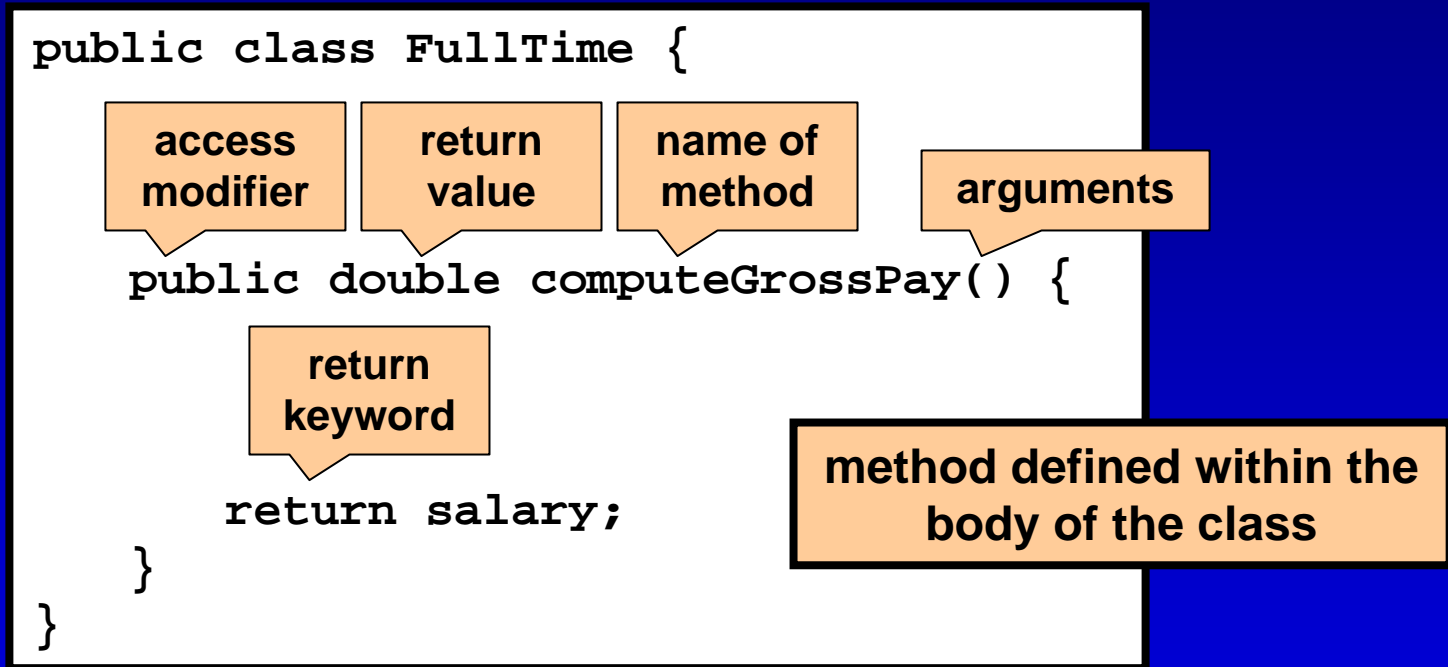
# Method Definition Syntax

- *[modifiers] returnType methodName(parameterType parameterName, parameterType parameterName)*
  - *returnType*
    - Type of value the method sends back to the method that calls it
    - Must be specified for all methods (with the exception of constructors)
    - If method returns no value, then its return type is void
    - If a method is declared to return a value, a return statement ***must*** be in the method body

# Methods Definition Syntax

- *[modifiers] returnType methodName(parameterType parameterName, parameterType parameterName)*
  - *methodName*
    - Name of the method
    - Can be any legal Java identifier
  - *(parameterType parameterName)*
    - Arguments of the method
    - Define what type of variables the method is going to use and the name of the variable
    - Multiple parameters are separated by a comma with type and name declared for each

# Method Definition Syntax



# More On Parameters

- When defining a method (in the body of a class), the parameters dictate the type and number of arguments passed when the function is invoked
- The names used for these parameters are arbitrary, as long as they are legal identifiers
- These variables, just like any other variable defined in a method, will be local to that method
- As soon as a method returns, any variables declared in that method are destroyed

# Signature of a Method

- When a method call is made, Java examines two properties to determine which method you are trying to use:
  - The identifier, which includes the class name and the method name
  - The number, type, and order of parameters
- These two properties comprise the *signature* of a method
- *Return type is not examined when determining signatures*

# Signature of a Method

```
FullTime emp01 = new FullTime();  
System.out.println(emp01.computeGrossPay());
```

examine the signature to determine the method to invoke

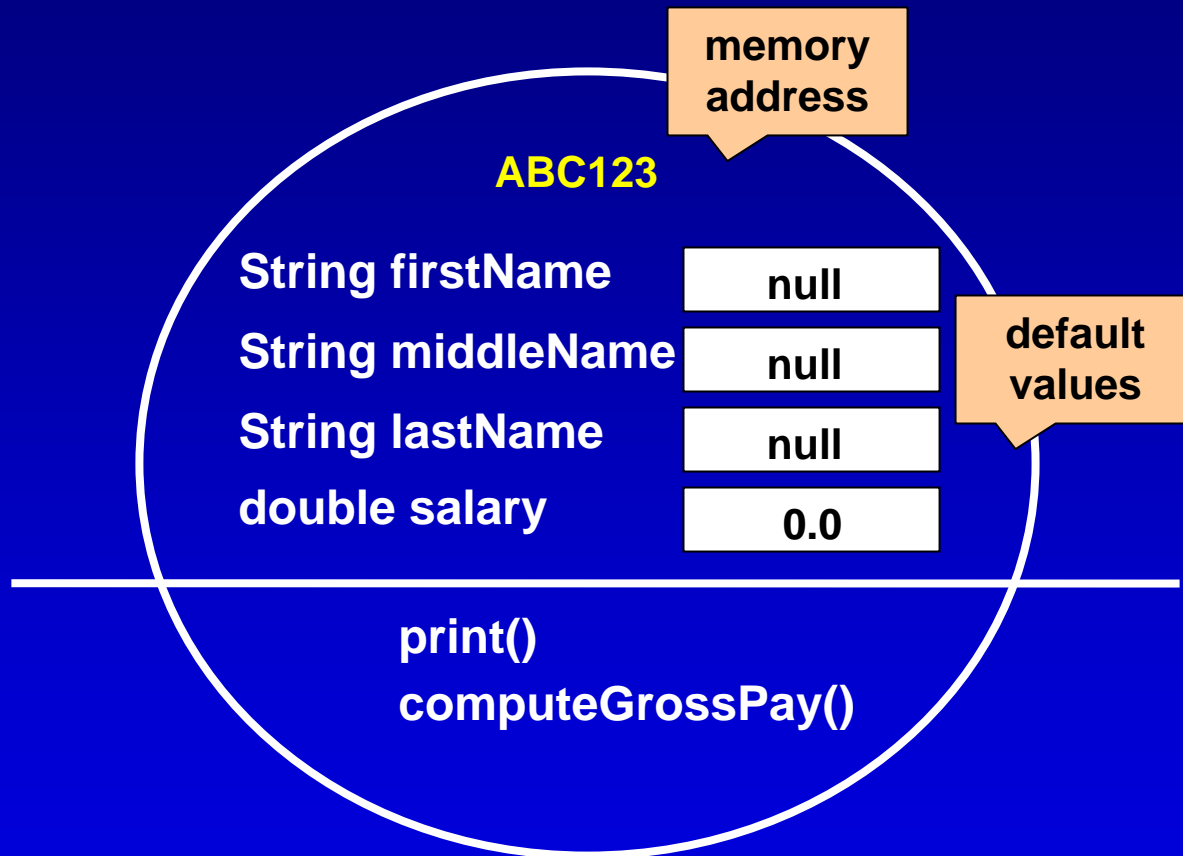
```
public class FullTime {  
    double salary;  
    . . . .  
    public double computeGrossPay() {  
        return salary;  
    }  
  
    public double computeGrossPay(int type) {  
        return salary * 1.02;  
    }  
}
```

method invoked – has matching signature

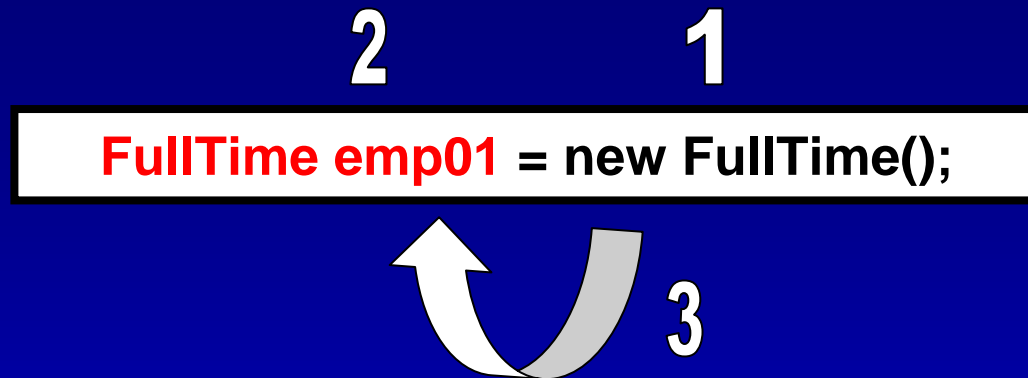
Not this method...



# Creating an Instance of a Class



# Creating an Instance of a Class



- 2
- Note : FullTime is a **user defined data type**. To define a variable, we use the syntax *type variableName*
  - A variable called emp01 is created of type FullTime
  - The variable emp01 is called **reference variable**

data type: FullTime



emp01

# Creating an Instance of a Class

- 3
- Assign the memory address to the variable emp01

data type: FullTime

ABC123

emp01

ABC123

String firstName	null
String middleName	null
String lastName	null
double salary	0.0

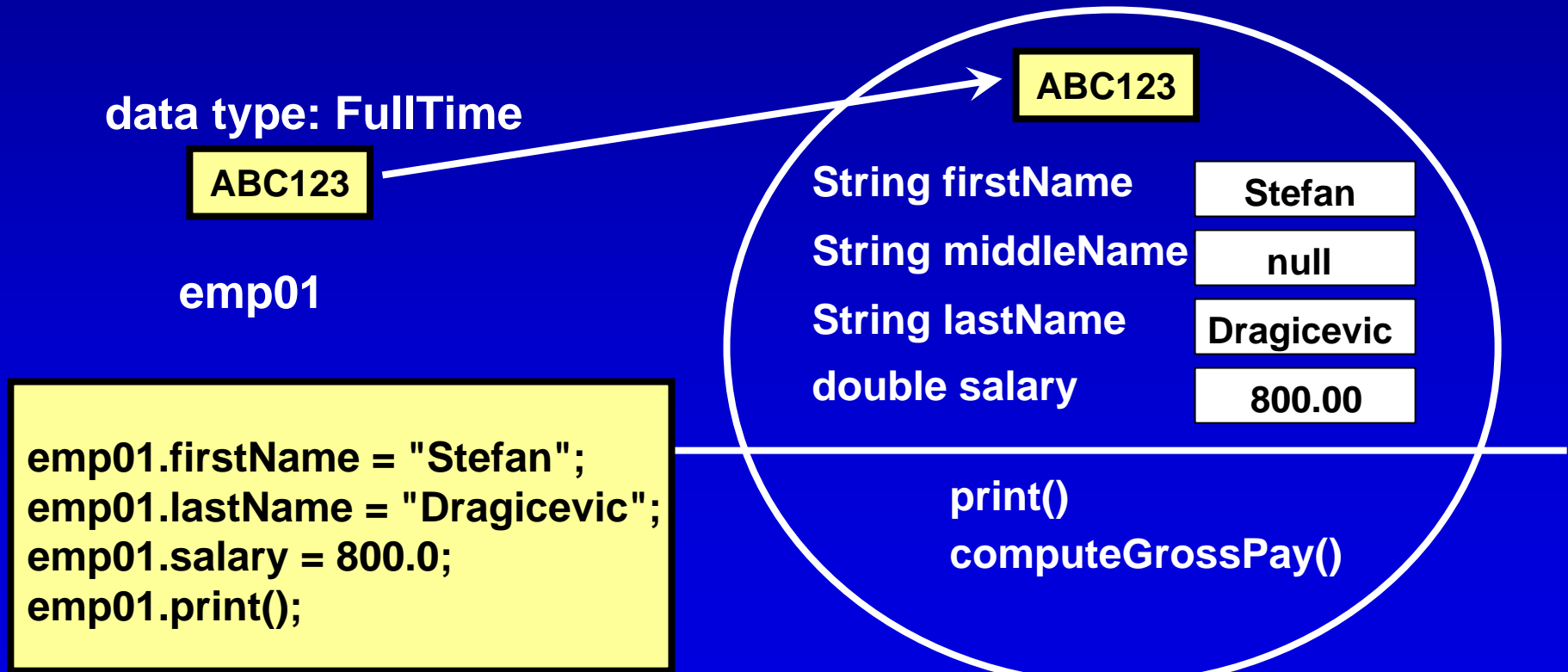
---

print()  
computeGrossPay()

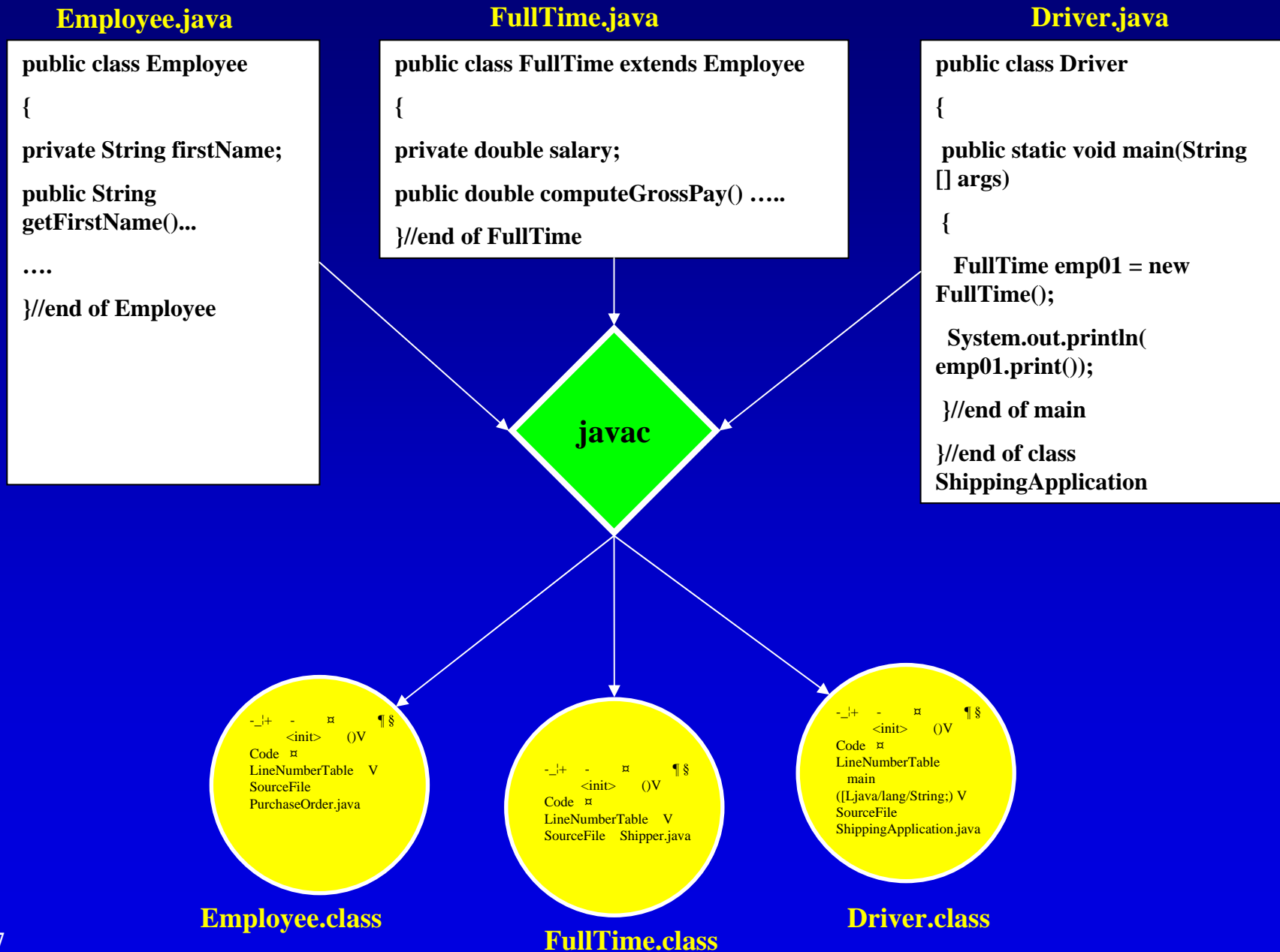
# Accessing Attributes and Methods

- To access the attributes or methods of the class, use the reference variable. The syntax will be:

```
objectName.attributeName;  
objectName.methodName();
```



# Java Development Cycle



# Hands-On Exercise

---

- Part 1 for Classes and Objects

# Scope of Variables

- Instance variable
  - Variables defined inside of a class but outside of a method
  - Available to every method in that class
- Local or automatic variable
  - Variable defined inside of a method are local to that method
- Class variable
  - More explanation in "More on Classes" chapter
- The scope of variables is a very important concept to understand and remember

# Constructors

- Constructors are specialized methods that only run when a new object is being created. They build and initialize objects
- Constructors have a very specific syntax
- If no constructors are defined in a class, the compiler inherits a no arguments constructor from the superclass
- If any constructors are defined, the default constructor will not be included

# Constructors

- Constructors are specialized methods that only run when a new object is being created
  - Use the new operator
  - Constructors cannot be called like regular methods
  - Constructors are automatically called by the JVM

```
FullTime emp01 = new FullTime();
```

# Constructor Declaration Syntax

- Constructors must have exactly the same name as the class they are contained in
- Constructors have no declared return type, not even the keyword void
- Constructor is automatically called via the new operator
- Constructors are usually public but in some cases are more restricted
  - The design pattern named Singleton uses a private constructor

# Constructor Declaration Syntax

- Sample constructor code below:

```
public class FullTime {  
    String firstName; String middleName;  
    String lastName; double salary;
```

Name of class == name of constructor

```
public FullTime( ) {  
  
}
```

Default constructor

no return value

```
public FullTime(String fName, String lName,  
    double sal) {  
    firstName = fName;  
    lastName = lName;  
    salary = sal;  
}
```

method overloading

# Defining Classes in a .java File

- The name of the class is the name of the java file

```
package com.protech.training;

import java.util.*;

public class FullTime {
    // body of the class
}
```

package statement

import statement

class declaration

- This file must be named **FullTime.java**

# Defining Classes in a .java File

- The .java file can contain three top-level elements:
  - *package* statement
  - *import* statement[s]
  - Class declaration[s]
- A .java file may contain only one public class declaration
- The file name must match the public class name, if included
- Remember, Java is case sensitive!

# Defining Classes in a .java File

```
package com.protech.training;

import java.util.*;

public class FullTime {
    String firstName;
    String middleName;
    double salary;

    public double computeGrossPay() {
        return salary;
    }
} //
```

Marks the start of the class declaration

Variable declaration

Method implementation

Marks the end of the FullTime class declaration

- The name of the class is FullTime
- The class modifier is the keyword public

# Reference vs. Value

- Java manipulates primitive data types by value and reference data types by reference to an object
- The == and != operators work a little differently for primitive and reference data types
  - For primitive data types, the value of the variable is compared
  - For reference data types, Java checks to see if both variables are referring to the same object

# Relational Operators

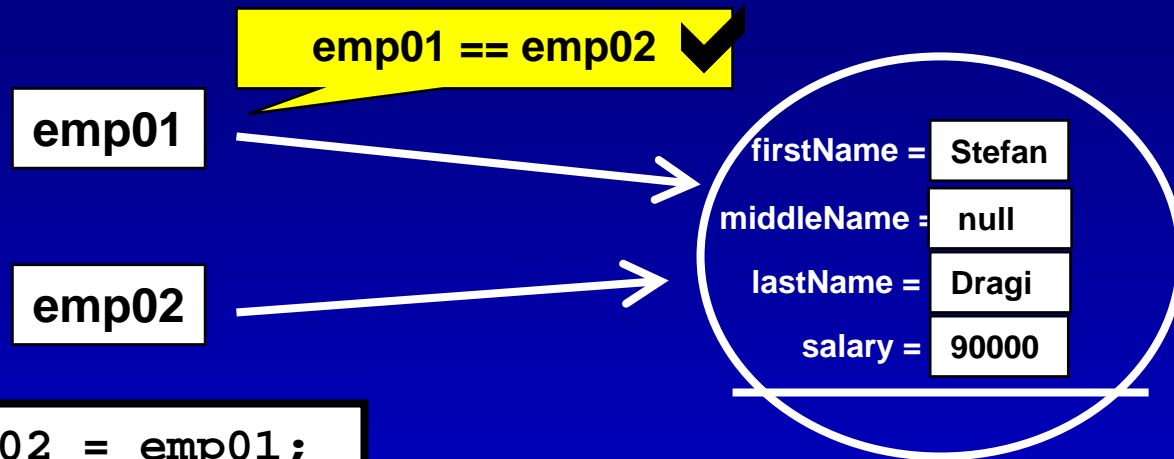
- If we had the following code section:

```
FullTime emp01 = new FullTime ("Stefan", "Dragi", 90000);  
FullTime emp02 = emp01;  
FullTime emp03 = new FullTime ("Stefan", "Dragi", 90000);
```

- emp01 == emp02 would return true, but
- emp01 == emp03 would return false
- Why?

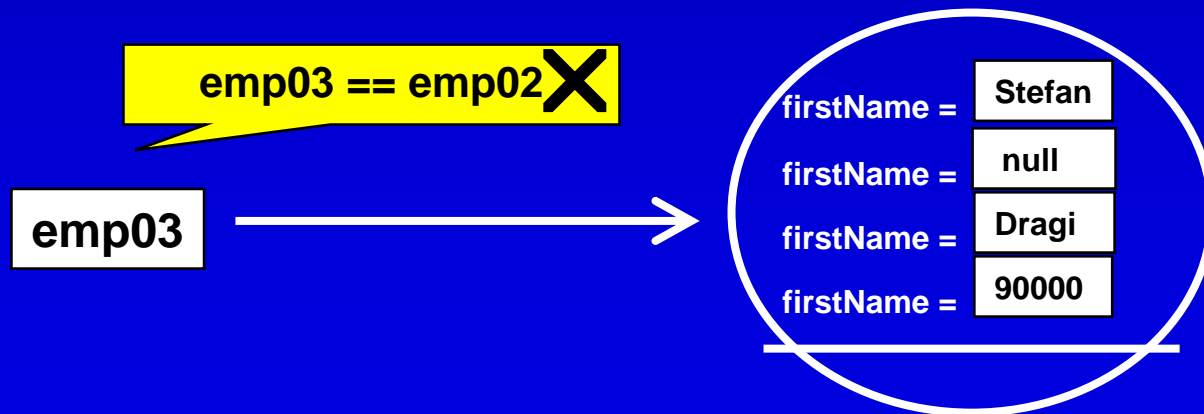
# == with Reference Variables

```
FullTime emp01 = new FullTime ("Stefan", "Dragi", 90000);
```



```
FullTime emp02 = emp01;
```

```
FullTime emp03 = new FullTime ("Stefan", "Dragi", 90000);
```



# Reference Variables

- More than one variable can refer to the same object, but any one variable may only refer to one object

data type: FullTime

ABC123

emp01

data type: FullTime

ABC123

emp02

ABC123

String firstName

null

String middleName

null

String lastName

null

double salary

0.0

print()

computeGrossPay()

```
FullTime emp01 = new FullTime( );  
FullTime emp02 = null;  
emp02 = emp01;
```

- Since both variables are of the same type and refer to the same object, they are synonymous to the JVM

# Reference Variables

- Assigning an object or *null* to a reference variable replaces whatever reference that variable was previously storing

data type: FullTime

ABC123

emp01

data type: FullTime

ABC123

emp02

ABC123

String firstName

null

String middleName

null

String lastName

null

double salary

0.0

print()

computeGrossPay()

```
FullTime emp01 = new FullTime( );  
FullTime emp02 = null;  
emp02 = emp01;  
emp01 = null;
```

*This slide intentionally left blank*