Software Development

4. Instantiable Classes.
How to Return a Value from a Method.
Getter and Setter Methods.
Constructors.

Revision

- How do we define the state/properties/attributes of a class of objects?
- How do we define the behavior of a class of objects?

Revision

- Our programs are built by using many classes
- We can see each class as a building block
  - e.g. Java Library classes: Scanner, System
  - We will write other building blocks (our classes) to perform tasks which are not provided by Java

Outline

- How to define instantiable classes
- How to return values from methods
- Setter and Getter Methods
- private and public access modifiers
- Variables Scope
- Constructors

Let's create a SimpleCalculator Application

- Develop an application to perform basic arithmetic operations on two numbers provided by the user
- We create and use an instantiable class
  - e.g. An existing instantiable class: Scanner
- The SimpleCalculator is our instantiable class, a **reusable block** that we create
- We separate the user's input and output from the computation (process)
- The instantiable class will do the processing/computation of the application

Let's create a SimpleCalculator Application

- Identify input, process, output

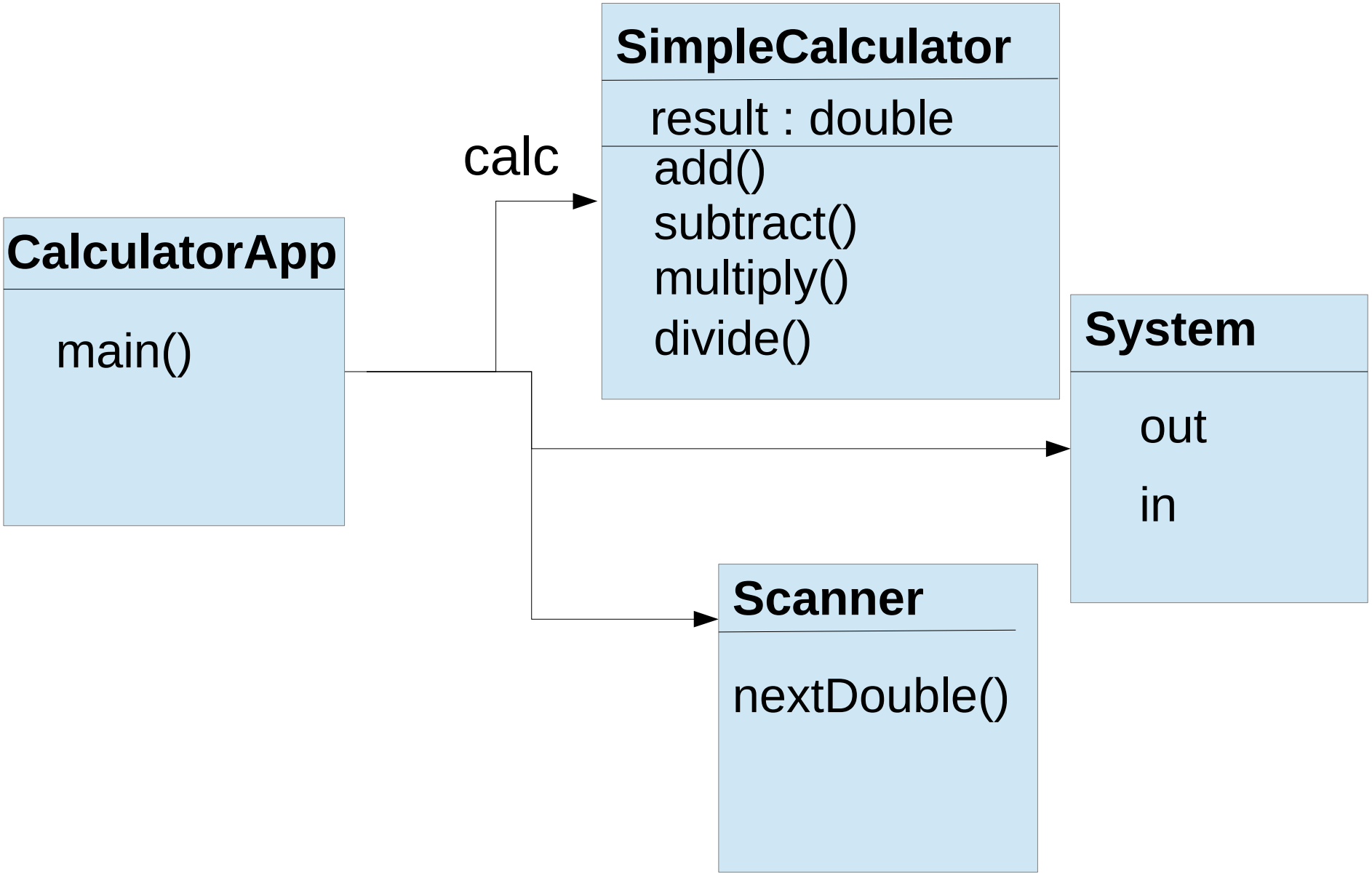
Let's create a SimpleCalculator Application

- Identify input, process, output
- Input
- Process
- Output

Let's create a SimpleCalculator Application

- Identify input, process, output
- Input
  - duble n1;
  - double n2;
- Process
  - The arithmetic operations: addition, subtraction, division, multiplication
- Output
  - double result;

Instantiable Classes



Instantiable Classes

- CalculatorApp.java
  - Contains the main() method
  - Did not create an instance/object of type CalculatorApp
  - Serve our own purposes, it cannot be reused by other programs
- SimpleCalculator.java
  - The class can be reused by many programs
    - How?

Instantiable Classes

- CalculatorApp.java
  - Contains the main() method
  - Did not create an instance/object of type CalculatorApp
  - Serve our own purposes, it cannot be reused by other programs
- SimpleCalculator.java
  - The class can be reused by many programs
    - How? **A:** by **creating**/ **instantiating** objects of type SimpleCalculator
  - An instantiable class (written in a different file) defines how the SimpleCalculator objects behave, and allows to create class instances/ objects of the SimpleCalculator data type

Methods

- Revision: general syntax to declare a method

```
<modifier> <returnDataType> <methodName>(<parameters>){
    // statements – code to represent the behaviour
}
```

Returning a Value From a Method

- What is unique to each method?

```
public void addTwoNumbers1(int a, int b){
    int sum = a + b;
    // print result
}
public int addTwoNumbers2(int a, int b){
    int sum = a + b;
    return sum;
}
```

Returning a Value From a Method

- What is unique to each method?

```
public void addTwoNumbers1(int a, int b){
    int sum = a + b;
    // print result
}
public int addTwoNumbers2(int a, int b){
    int sum = a + b;
    return sum;
}
```

Method's return data type

Returning a Value From a Method

- What is unique to each method?

```
public void addTwoNumbers1(int a, int b){
    int sum = a + b;
    // print result
}
public int addTwoNumbers2(int a, int b){
    int sum = a + b;
    return sum;
}
```

Method's return data type

Data type of the return value

Returning a Value From a Method

- What is unique to each method?

```
public void addTwoNumbers1(int a, int b){
    int sum = a + b;
    // print result
}
```

A void method does not return any value!

```
public int addTwoNumbers2(int a, int b){
    int sum = a + b;
    return sum;
}
```

The data type of the return value **must** match the method's return type!

Returning a Value From a Method

- We used the keyword return followed by the value/ expression a method must return
- The return statement is the last statement in a method's body
- The data type of the return value must match the method's return data type

Setter and Getter Methods

- e.g.

```
public class SimpleCalculator {
    private double firstNumber; // declare an instance variable
    // more code goes here
}
```

- The private instance variable firstNumber can be accessed by the methods from the SimpleCalculator class, but not by the methods from the CalculatorApp class
- To allow other classes to have access to the private member variables we can use **public** Setter and Getter Methods

Setter Methods

- A method which sets/mutates the value of a member variable is called a **setter method**
- A setter method **does not return any value**
- By convention, the name of a setter method is prefixed by the word **set**
- e.g.

```
public void setFirstNumber(double number){
        firstNumber = number; /* sets/ assigns the value provided via the parameter number  to the instance variable firstNumber */
}
```

Getter Methods

- A method which gets/retrieves the value of a member variable is called a **getter method**
- A getter method **must always return a value**; the getter method's return type is the data type of the value it returns
- A getter method **does not take in any parameters**
- By convention, the name of a getter method is prefixed by the word **get**
- e.g.

```
public double getResult(){
  return result;  //returns the value contained in the result instance variable
}
```

Access Modifiers

- public
  - Variables or methods declared with a public access modifier can be accessed/used by any methods (declared either in the same class or in a different class)
    - e.g. If add() is public it can be used within CalculatorApp class
- private
  - Variables or methods declared with a private access modifier can be accessed/used only by the methods which are declared in the same class with them
    - e.g. If add() is private it cannot be used within CalculatorApp class

Instance Variables

- Instance variables are declared inside a class declaration, but outside of a method's body
- <modifier> <dataType> <fieldName>;
- <modifier>
  - public, private, protected
  - In general, we should use the private modifier when we do not want to allow others to change the state/attributes of an object – only the methods defined within the same class they are declared in can access them
- e.g. public class SimpleCalculator { private double firstNumber; //...}
  - The instance variable firstNumber can be accessed/used by the methods from the SimpleCalculator class, but not by the methods from the CalculatorApp class

Variables Scope

- Local variables
  - Declared within the body of a method
  - Can be accessed only within the body of the method in which they are declared (that is, they are not known outside of that method)
    - e.g. Recall that we declared a Scanner sc variable in the main() method, we cannot use sc variable outside the main()
- Member variables
  - private variables
    - can be accessed within any method declared in the same class with them
  - public variables
    - can be accessed within any methods (declared either in the same class or in a different class)

Constructors

- Creating/Instantiating an object

  e.g. SimpleCalculator calc = new SimpleCalculator();

new creates a new object of the specified class

The type of the object is SimpleCalculator

calc is a variable, the identifier/name to refer the new object

Constructor is used to initialize the new object's instance variables when the object is created

- A constructor is implicitly called by the new operator during the object construction

Constructors

- Creating/Instantiating an object

  e.g. SimpleCalculator calc = new SimpleCalculator();

Constructors

- Creating/Instantiating an object

  e.g. SimpleCalculator calc = new SimpleCalculator();

  – SimpleCalculator calc;

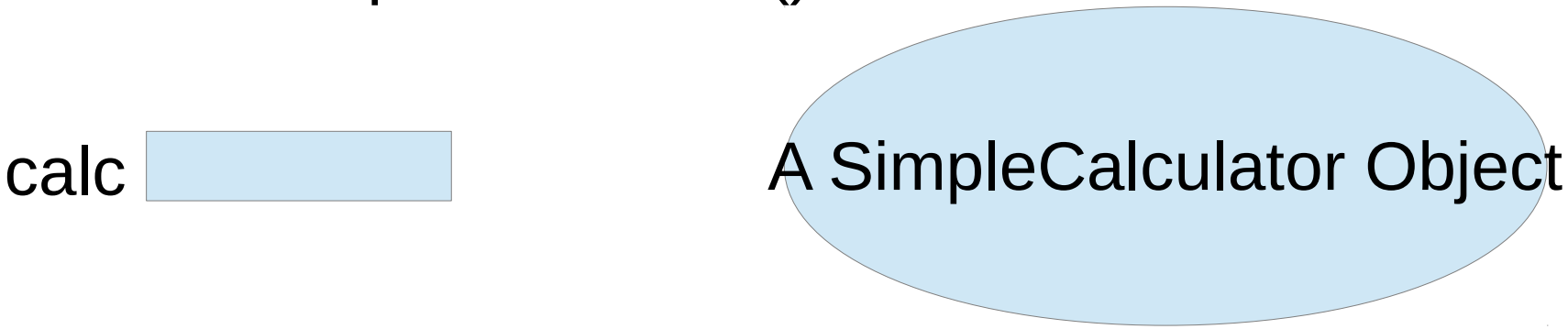
calc 

Constructors

- Creating/Instantiating an object

  e.g. SimpleCalculator calc = new SimpleCalculator();

  - new SimpleCalculator();

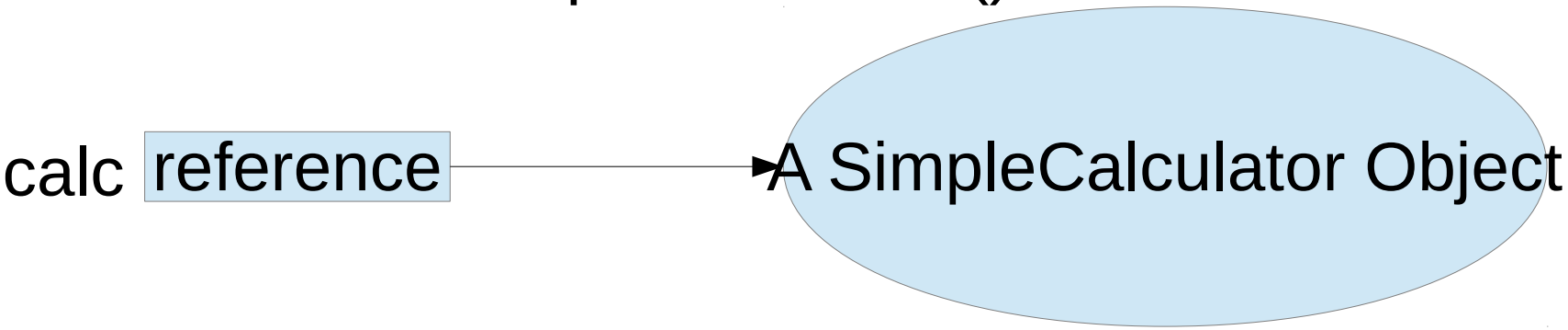


Constructors

- Creating/Instantiating an object

  e.g. SimpleCalculator calc = new SimpleCalculator();

  - calc **=** new SimpleCalculator();



Constructors

- Creating/Instantiating an object

  e.g. SimpleCalculator calc = new SimpleCalculator();

- calc = **new** SimpleCalculator();
  - **creates an instance** of the class SimpleCalculator
  - returns a **reference** to the new object
  - the reference to the new object is stored, in this example, in the variable calc

Constructors

- The constructors declaration looks like a method, except they must have **the same name as the class** and **do not have a return type**
- Declaring a constructor

```
<modifier> <ClassName>(<parameters>)
{      /* code */ }

<modifier> <ClassName>() { /* code */ }
```

- Usually, the <modifier> is public

Default Constructor

- Last week, we were able to create objects of type Calculator even though the Calculator class did not contain any constructor declaration in that file
- Java compiler, provides a **default constructor**, that is a constructor which has no parameters
- If we use the default constructor, the instance variables declared in that class are intialized to their **default values**

Member Variables – Default Values

- The default variables for primitive member variables:
  - byte ==> (byte) 0
  - short ==> (short) 0
  - int ==> 0
  - long ==> 0L
  - float ==> 0.0f
  - double ==> 0.0d
  - boolean ==> false
  - char ==> '\u0000' (i.e. the null character)
- All Reference types (so far we know the class type which is a subset of the reference type):
  - e.g. SimpleCalculator calc; String myText;
  - Are intialised by default to null - value which shows that there is no reference to an object yet (i.e. the object of that particular type has not been created yet)

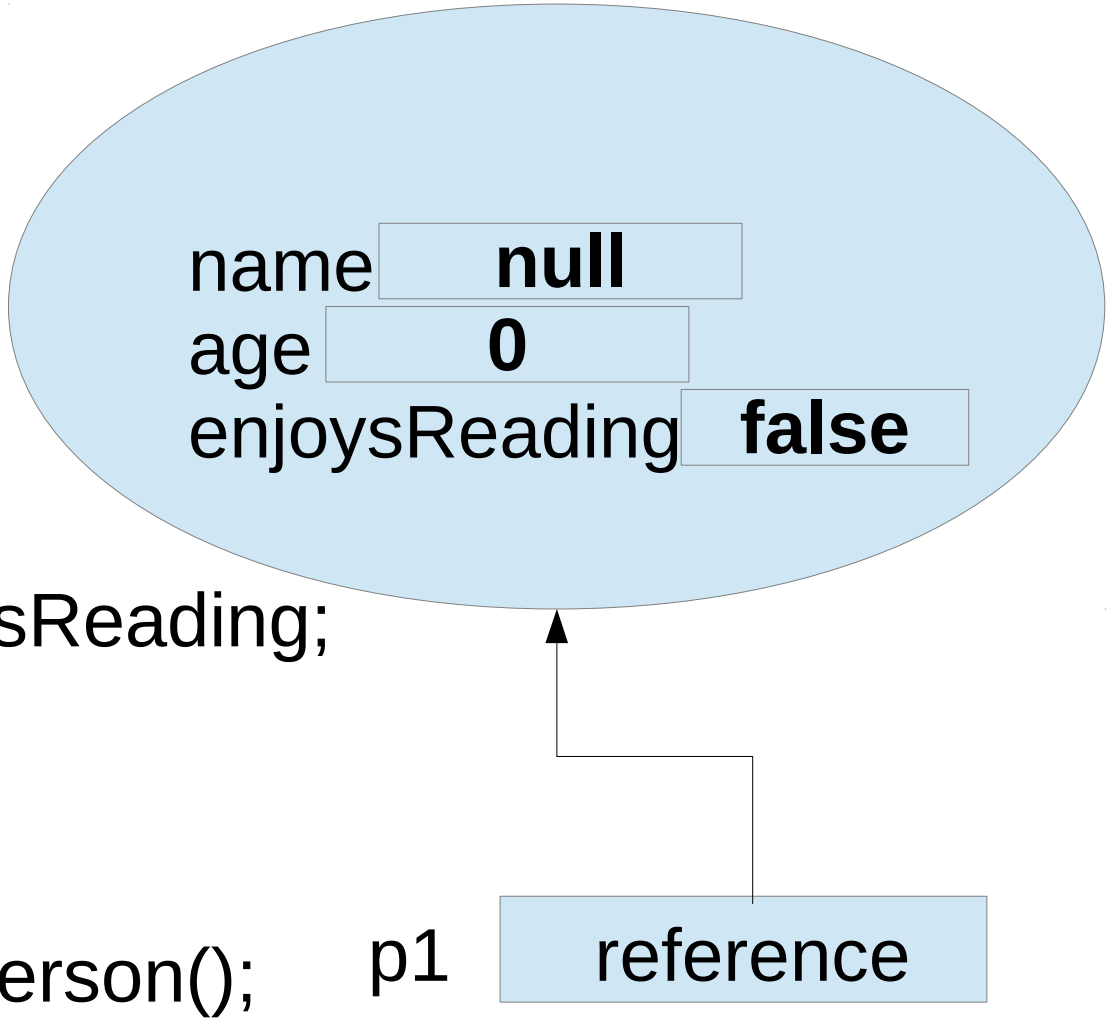
Default Constructor

```
public class Person{
   // instance variables
   private String name;
   private int age;
   private boolean enjoysReading;
   // some methods
}
```

Default Constructor

```
public class Person{
   // instance variables
   private String name;
   private int age;
   private boolean enjoysReading;
   // some methods
}
```

e.g. Person p1 = new Person();



Constructors with Parameters

- Typically, we should customize the initialization to make sure that the instance variables are initialized with meaningful values
  - Solution: declare and use a constructor with parameters
- Declaring a constructor

```
<modifier> <ClassName>(<parameters>) {
/* code to initialize the instance variables */
}
```

Constructors with Parameters

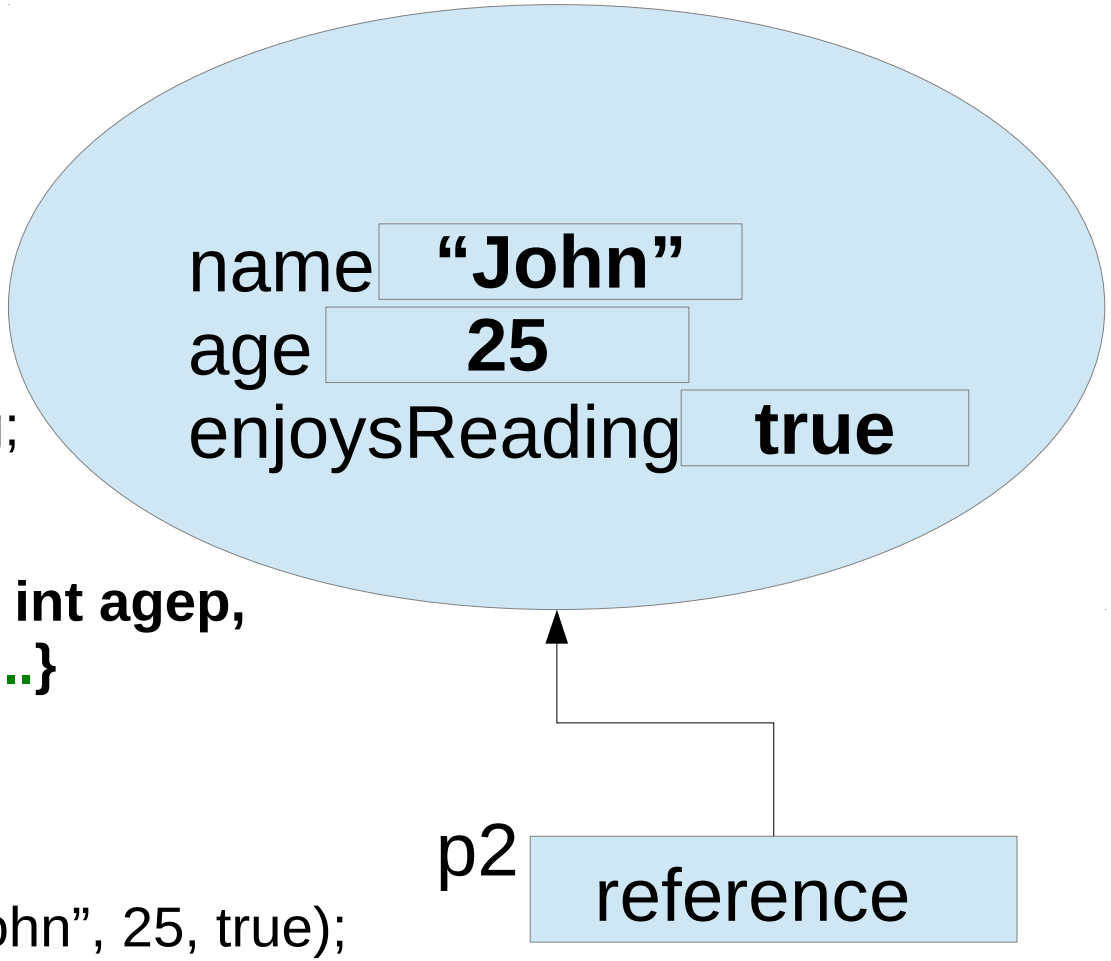
- e.g.

```
public Person(String namep, int agep,
                    boolean lovesReading){
   name = namep;
   age = agep; // stores the value from agep in age
   enjoysReading = lovesReading;
}
```

Constructors with Parameters

```
public class Person{
   // instance variables
   private String name;
   private int age;
   private boolean enjoysReading;
   // constructor with parameters
   public Person(String namep, int agep,
boolean lovesReading){//code...}
   // some methods
}
```

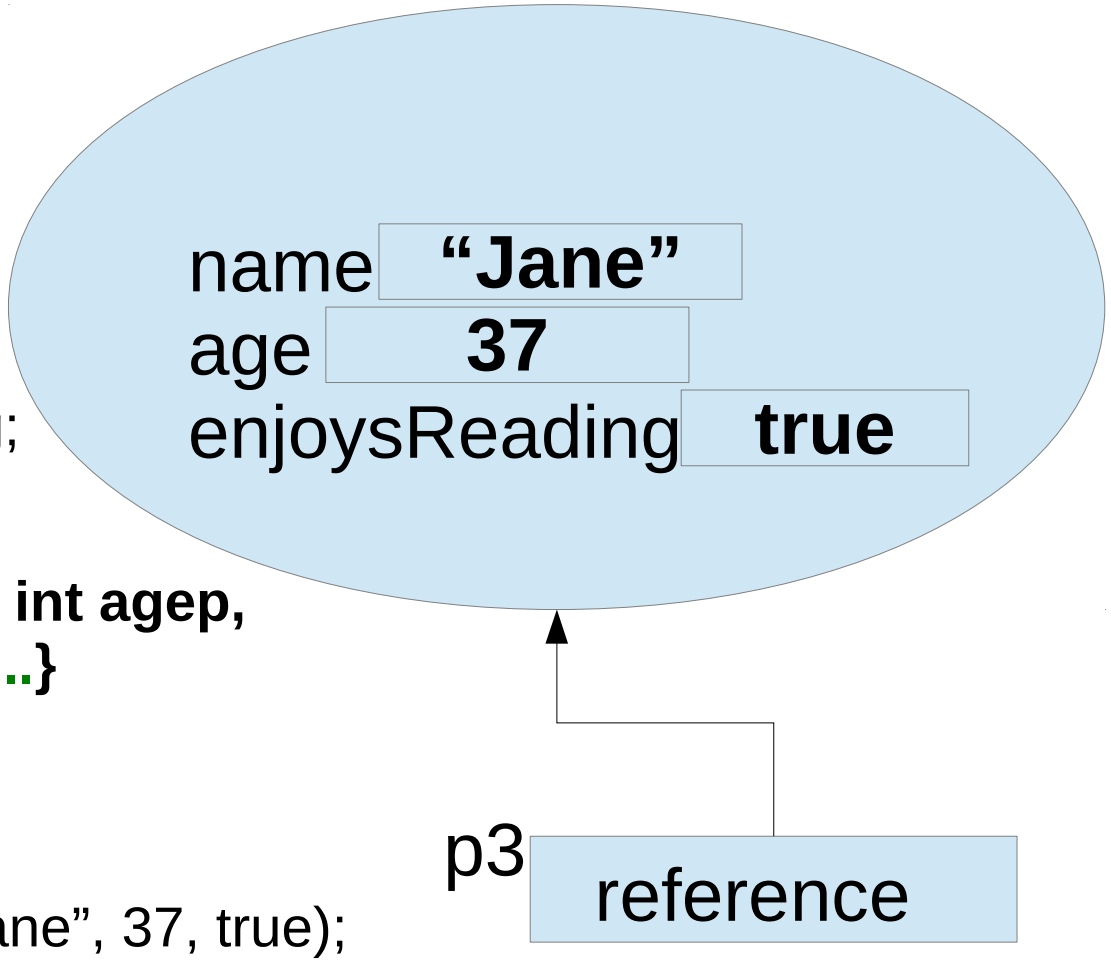
e.g. Person p2 = new Person(“John”, 25, true);



Constructors with Parameters

```
public class Person{
   // instance variables
   private String name;
   private int age;
   private boolean enjoysReading;
   // constructor with parameters
   public Person(String namep, int agep,
boolean lovesReading){//code...}
   // some methods
}
```

e.g. Person p3 = new Person(“Jane”, 37, true);



Constructors

- If you declare a constructor with parameter(s) in a class, then the Java compiler **will not create** the default constructor for that class any more
- e.g. In the Person example, if we declared only the constructor with parameters then we cannot create an object using the constructor with no parameters
  - ~~e.g. Person p4 = new Person();~~ // compilation error

Constructors

- If you declare a constructor with parameter(s) in a class, then the Java compiler **will not create** the default constructor for that class any more
- If we want to create objects using the constructor without parameters, then we also have to declare a constructor which takes no arguments
- e.g. public Person(){ /*code, if any*/}
  - e.g. Person p4 = new Person(); // now this compiles

Constructors with Parameters

- e.g.

```
public Person(String name, int age,
    boolean enjoysReading){
        // name = name; // it is not correct
        // age = age; // it is not correct
        // enjoysReading = enjoysReading; // it is not correct
}
```

- Each parameter shadows one of the instance variables

Constructors with Parameters

- e.g.

```
public Person(String name, int age,
    boolean enjoysReading){
        // name = name; // it is not correct
        // age = age; // it is not correct
        // enjoysReading = enjoysReading; // it is not correct
}
```

- Inside the constructor, **name** is a local copy of the first argument passed to the constructor

Constructors with Parameters

- e.g.

```
public Person(String name, int age,
    boolean enjoysReading){
        this.name = name;
        this.age = age;
        this.enjoysReading = enjoysReading;
}
```

- **this.name** refers to the instance variable called name of the object which is currently initialized

this

- **this –** is a **reference** to the current object, namely the object whose constructor or method is currently being called/ invoked
  - this – is a Java keyword
- We can use this within any object's method or a constructor to access/refer to any instance variable or object's (instance) method

Multiple Constructors

- We can have multiple scenarious to initialize an object
  - e.g. We would like to allow an object of type Person to be created also when an individual does not want to reveal his/her age
- We can declare multiple constructors for a single class, this is called overloading constructors

Multiple Constructors

- We overload constructors by declaring multiple constructors with different signatures
- The parameters list **must** have different number of parameters with different data types and different order of the parameters data types

Multiple Constructors – Example

- public Person(String name, int age,
  boolean enjoysReading){

  this.name = name;

  this.age = age;

  this.enjoysReading = enjoysReading;

  }

Multiple Constructors – Example

- public Person(){

  this.name = null;

  this.age = 0;

  this.enjoysReading = false;

  }
- // or, the equivalent of the above is the following
- public Person() { }

Multiple Constructors – Example

- public Person(String name) {

  this.name = name;

  }

Multiple Constructors – Example

- public Person(String name, int age){

  this.name = name;

  this.age = age;

  }

Multiple Constructors – Example

- public Person(String name, boolean enjoysReading){

```
    this.name =  name;
    this.enjoysReading = enjoysReading;
}
```

Summary: SimpleCalculator

- Let's compare the SimpleCalculator.java with the Calculator.java (available on moodle, under Topic 3)
  - Declared the two numbers as instance variables
  - Declared constructor(s)
  - Created setter and getter methods
- Declared another class, CalculatorApp, which **reuses** the **instantiable** SimpleCalculator class

Resources

- Java Language Keywords
  - http://docs.oracle.com/javase/tutorial/java/nutsandbolts/