Programming using Java

Java Classes and Objects: A Preview

Static Initialization Statement

- The Statement to be executed at the same time when the system initialize the static variable in the class

- From

static { <statement> **}**

Static Initialization Statements

```
public class Test {
    static{
        System.out.println("Static");
    }
    {
        System.out.println("Non-static block");
    }
    public static void main(String[] args) {
        Test t = new Test();
        Test t2 = new Test();
    }
}
```

Static
Non-static block
Non-static block

Static Initialization Statement

- Execution Order
  - Order of initialization of static init. Statement and static variable : existing order in the program

```
class Initializers {
        static {   i = j + 2; }        // Error
        static int i, j;
        static j = 4;
        //...
}
```

  - Executed earlier than constructor

finalize Method

- Garbage Collector
  - Automatic Memory Management

- finalize Method
  - Call the finalize method before the garbage collector reclaim the memory
- Provide the method to release the resources
  - Programmer can remove the resources(ex:open files) directly using finalize method which garbage collector cannot reclaim.

Fraction Methods

```
public class Fraction
{ int num, denom;
  public Fraction (int p, int q)
  {
     num = p;
     denom = q;
  }
```

```
class FractionDemo {
  public static void main(String args[]) {
    //Create 3 Fraction objects,
// initialize 2 with 4 hardcoded values (i,j,k,l)
   // Print them on screen
   // add the two Fraction objects and assign it to third Fraction
 }
}
```

```
  public add ( Fraction f)…
  public Fraction invert (Fraction f)…
}
```

return

- A method, unless void, returns a value of the specified type to the calling method.
- The return statement is used to immediately quit the method and return a value:

```
return expression;
```

The type of the return value or expression must match the method's declared return type.

return

- A method can have several return statements; then all but one of them must be inside an if or else (or in a switch):

```
public someType myMethod (...)
{
    ...
    if (...)
        return <expression1>;
    else
        return <expression2>;
    ...
    return <expression3>;
}
```

return

- A boolean method can return true, false, or the result of a boolean expression:

```
public boolean myMethod
(...)
{
   ...
   if (...)
      return true;
   ...
   return n % 2 == 0;
}
```

return

- A void method can use a return statement to quit the method early:

```
public void myMethod (...)
{
    ...
    if (...)
        return;
    ...
}
```

No need for a redundant return at the end

return

- If its return type is a class, the method returns a reference to an object (or null).
- Often the returned object is created in the method using new. For example:

```
public Fraction inverse ()
{
    if (num == 0)
        return null;
    return new Fraction (denom, num);
}
```

- The returned object can also come from the arguments or from calls to other methods.

The main Method

The Main Method - Concept

- **main** method
  - the system locates and runs the main method for a class when you run a program
  - other methods get execution when called by the main method explicitly or implicitly
  - must be public, static and void

The Main Method - Getting Input from the Command Line

- When running a program through the `java` command, you can provide a list of strings as the real arguments for the `main` method. In the `main` method, you can use `args[index]` to fetch the corresponding argument

```
class Greetings {
  public static void main (String args[]){
    String name1 = args[0];
    String name2 = args[1];
    System.out.println("Hello " + name1 + "&" +name2);
  }
}
```

```
➢ java Greetings Jacky Mary
Hello Jacky & Mary
```

- Note: What you get are strings! You have to convert them into other types when needed.

Passing Arguments

- Primitive data types are always passed "by value": the value is copied into the parameter

```
public class MyMath
{
   ...
   public double square (double x)
   {
      x *= x;
      return x;
   }
}
```

x here is a copy of the argument. The copy is changed, but...

```
MyMath calc = new MyMath();
double x = 3.0;
double y = calc.square (x);
System.out.println (x + "  " + y);
...
```

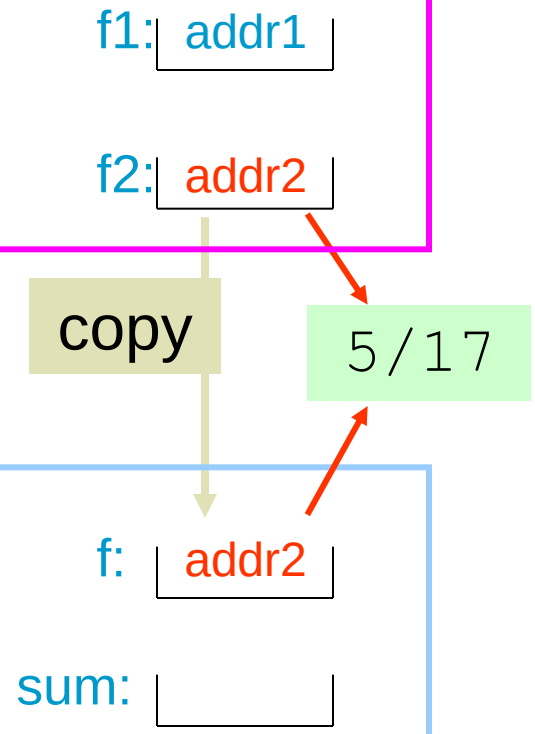
... the original x is unchanged. Output: 3 9

Passing Objects as Arguments

- Objects are always passed as references: the <u>address</u> is copied, not the object.

```
Fraction  f1 = new Fraction (1, 2);
Fraction  f2 = new Fraction (5, 17);
Fraction  f3 = f1.add (f2);
```

```
public class Fraction
{
   ...
   public Fraction add (Fraction f)
   {
      Fraction sum;
            ...
   }
}
```



Libraries

- Java programs are usually not written from scratch.
- There are hundreds of library classes for all occasions.
- Library classes are organized into packages. For example:

  java.util — miscellaneous utility classes
  java.awt — windowing and graphics toolkit
  javax.swing — newer GUI package Swing

Programmers write classes

- And extensively use library classes
  - either directly:

```
JButton go = new JButton("Click here");
```

  - or through inheritance:

```
public class LetterPanel extends JPanel
```

import

- Full library class names include the package name. For example:

  java.awt.Color
  javax.swing.JButton

- import statements at the top of your program let you refer to library classes by their short names:

```
import javax.swing.JButton;
...
    JButton go = new JButton("Click here");
```

Fully-qualified name

import (cont'd)

- You can import names for all the classes in a package by using a wildcard `.*`:

```
import   java.awt.*;
import   java.awt.event.*;
import   javax.swing.*;
```

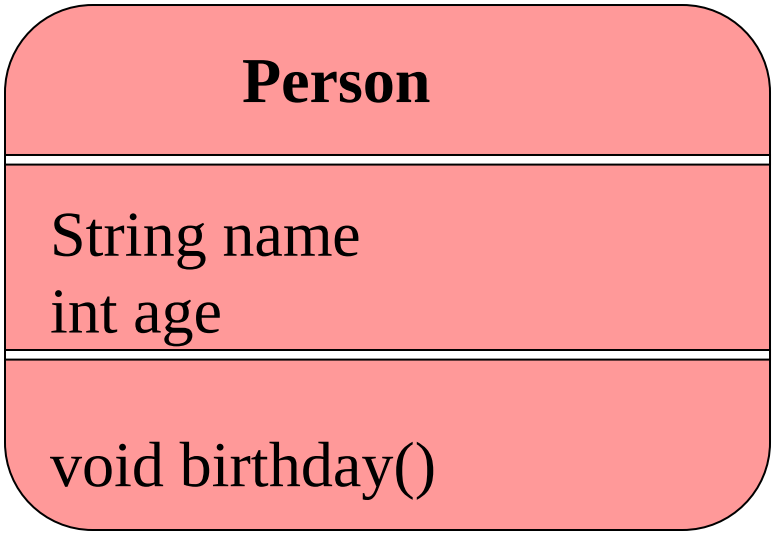
Imports all classes from awt, awt.event, and swing packages

- java.lang is imported automatically into all classes; defines System, Math, Object, String, and other commonly used classes.

An example of a class

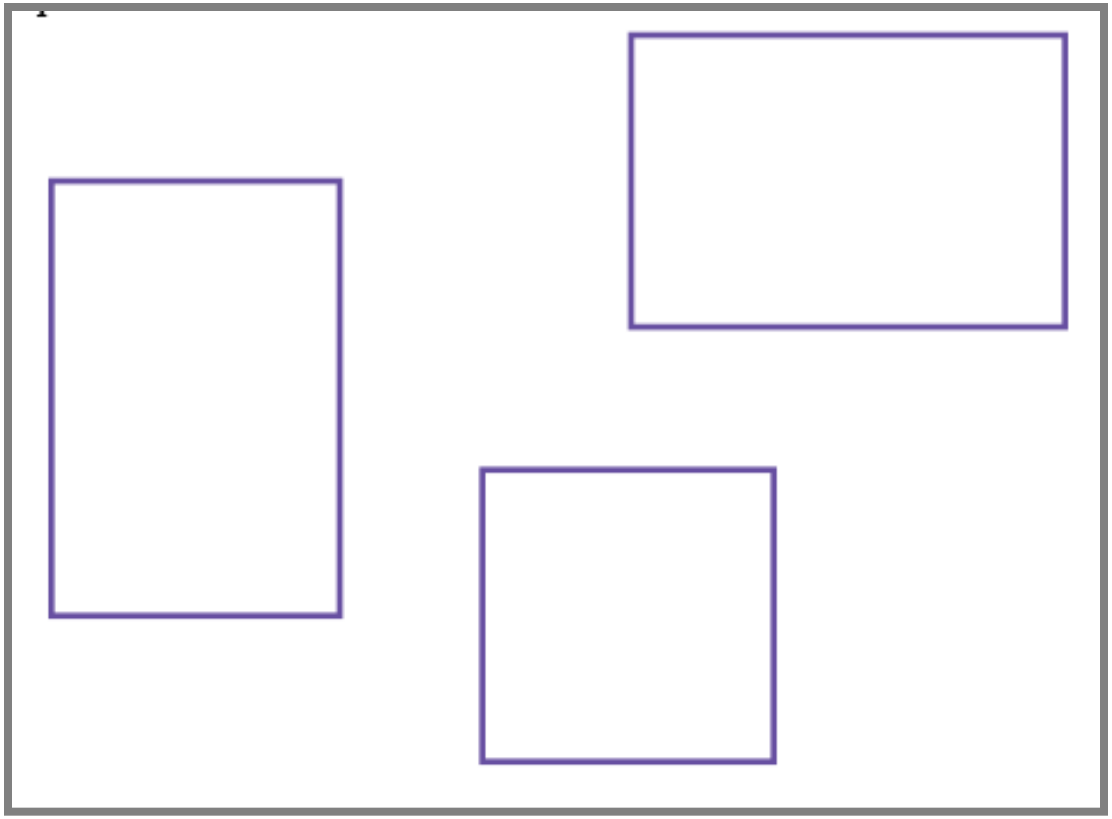
Create a class HelloWorldPerson which will:

- Create 5 person objects and add to a Person array (hard code values)
- Print as output the names of all the Persons in the array
- Sort the array in increasing order of the ages of Person



Rectangular Shapes and Rectangle Objects

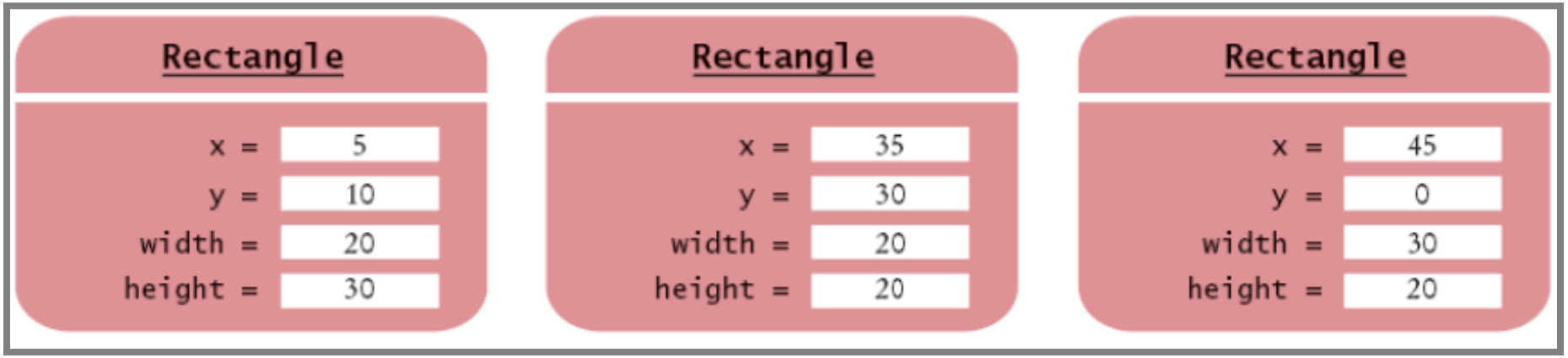
- Objects of type `Rectangle` *describe* rectangular shapes



Rectangular Shapes

Rectangular Shapes and Rectangle Objects

- A `Rectangle` object isn't a rectangular shape–it is an object that contains a set of numbers that describe the rectangle



Rectangular Objects

Rectangle Class

Give code for Rectangle class

- It should have 4 fields of int type for x, y, width and height
- It should have 2 constructors

1. No argument constructor which sets all field values to 0
2. Four argument constructor which sets all field values.

- Translate method: which will move a Rectangle to the coordinates passed as arguments.

Give code for a RectangleDemo class

- Create Four Rectangle objects
- Identify if any of them overlap !!!!!

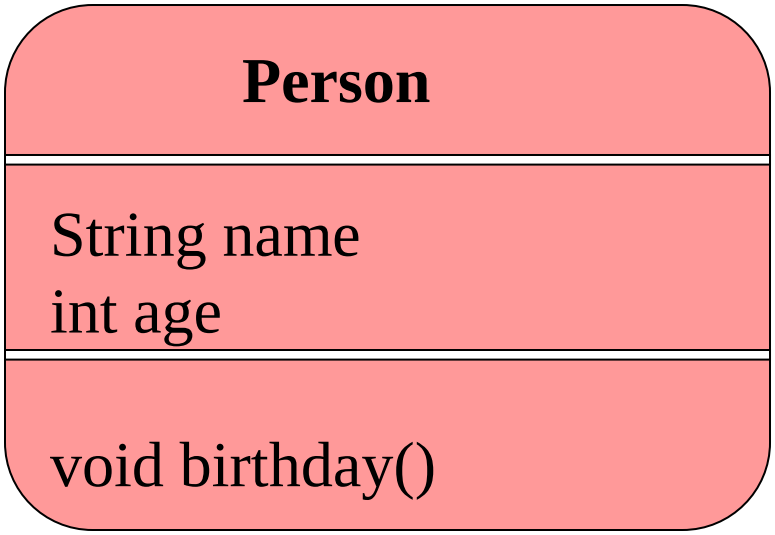
Constructing Objects

- ```
  new Rectangle(5, 10, 20, 30)
  ```
- Detail:
  1. The new operator makes a `Rectangle` object
  2. It uses the parameters (in this case, `5`, `10`, `20`, and `30`) to initialize the data of the object
  3. It returns the object
- Usually the output of the new operator is stored in a variable

```
Rectangle box = new Rectangle(5, 10, 20, 30);
```
```

Constructing Objects

- The process of creating a new object is called *construction*
- The four values `5`, `10`, `20`, and `30` are called the *construction parameters*
- Some classes let you construct objects in multiple ways

```
new Rectangle()
// constructs a rectangle with its top-left corner
// at the origin (0, 0), width 0, and height 0
```

Self Check

- How do you construct a square with center (100, 100) and side length 20?

Answers

- ```
  new Rectangle(90, 90, 20, 20)
  ```
  ```

An example of a class

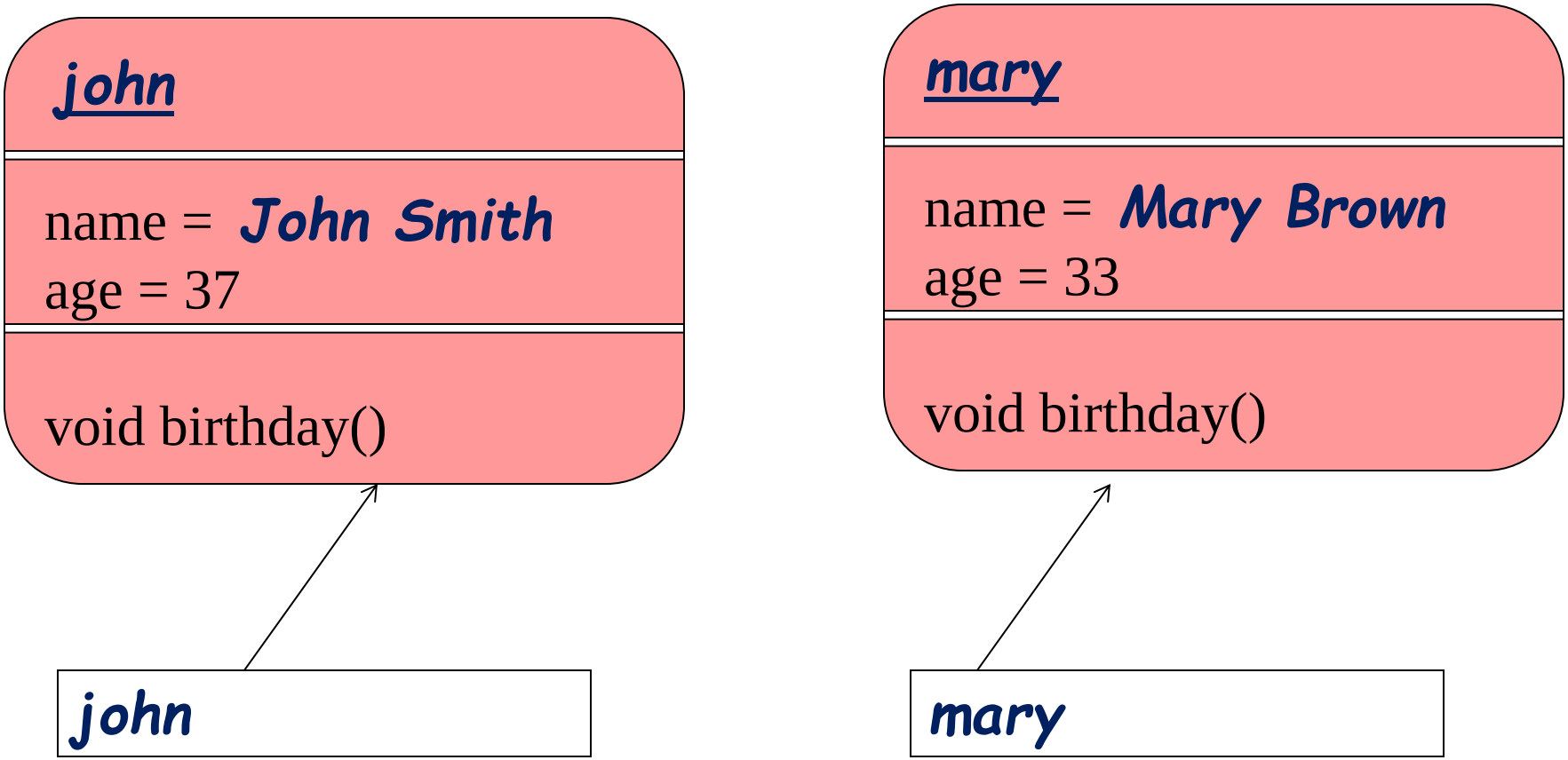
```
class Person {
    String name;
    int age;

    void birthday ( ) {
        age++;
        System.out.println(name+ 'is now '

            +age);
    }
}
```



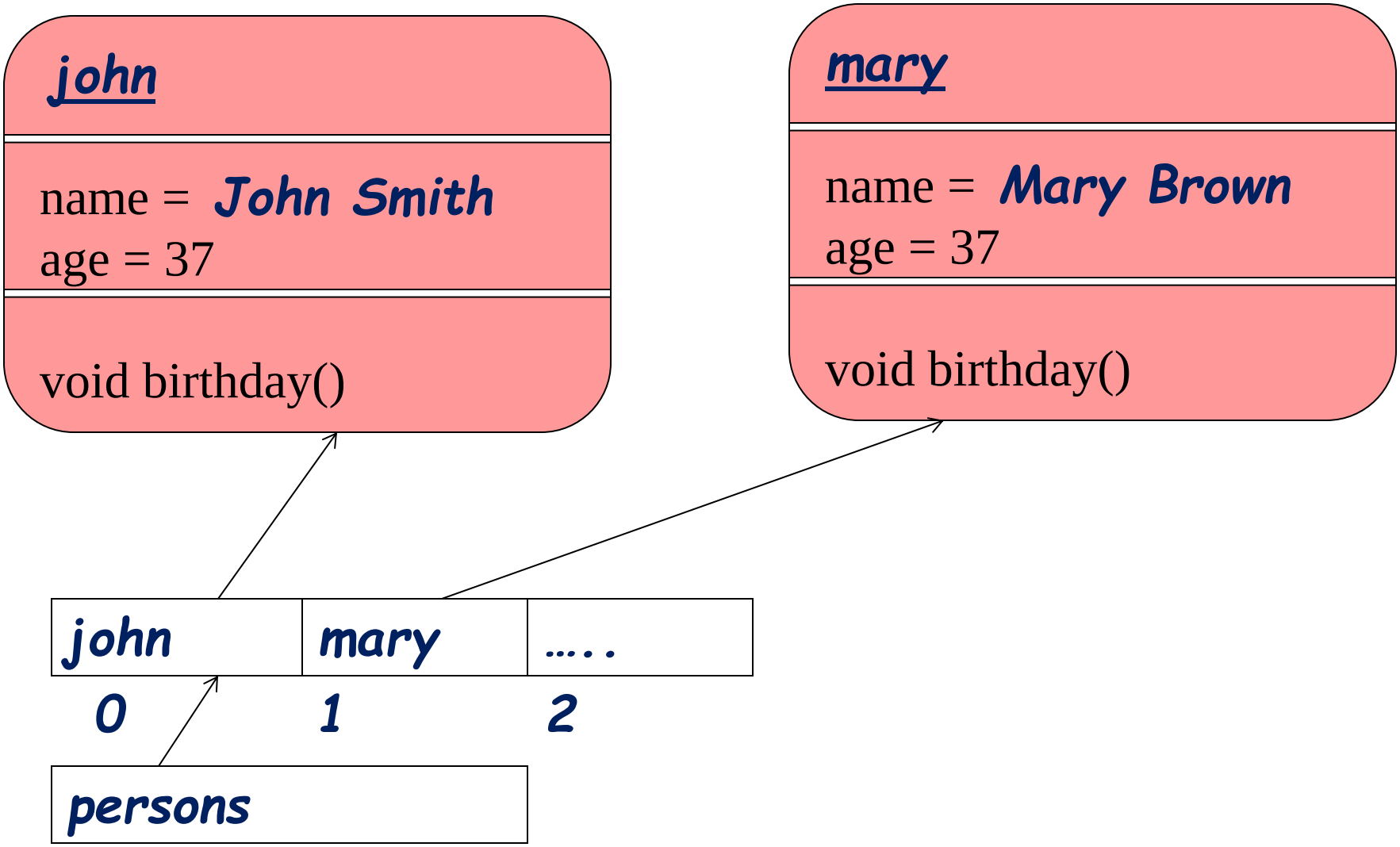
Creating and using objects

```
public class HelloWorldPerson {
   public static void main(String[] args){
     Person john;                    // Declaring object
      john = new Person ( );  // creating object
      john.name = "John Smith";
      john.age = 37;


     Person mary = new Person ( );
      mary.name = "Mary Brown";
      mary.age = 33;
      mary.birthday ( );
   }
 }
```

Examples of Objects of type Person



Array of Objects of type Person



Array of Objects of type Person

```
Person [] persons= new Person[10];
persons[0] = john;
….
```