Java Lec 2_Classes

Monday, August 23, 2021 8:18 PM

Java_Lec_...

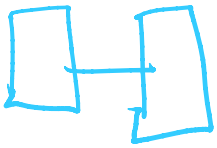
Programming using Java

Our own Classes
-create objects.

Java Classes and Objects

Objectives:

- Get an introduction to Classes and Objects
- Get a general idea of how a small program is put together



Classes and Source Files

- A class defines a class of objects.

Convention:
a class name starts with a capital letter

Class name:	File name:
SomeClass	SomeClass.java
Ramblecs	Ramblecs.java
FallingCube	FallingCube.java

Same upper / lower case letters

Files and Folders

Compilation

- javac automatically looks for classes (.java or .class files) in the current folder, or, if classpath is set, in folders listed in the classpath string.

IDE

- A missing file may be reported as a syntax error when compiling another file.
- If you set classpath, include the current folder. It is denoted by a dot. For example:

  .;C:\javamethods\EasyIO

- IDE helps take care of the file locations.

public class *SomeClass*

{

- *Fields* — Attributes / variables that define the object's state. Can hold numbers, characters, strings, other objects. Usually private.
- *Constructors* — Code for constructing a new object and initializing its fields. Usually public.
- *Methods* — Actions that an object can take. Can be public or private.

}

style

private: visible only inside this class
public: visible in other classes

```
public class FallingCube
{
  private final int cubeSize;
  private int cubeX, cubeY;    // Cube coordinates
  ...
  private char randomLetter;   // Cube letter

  public FallingCube(int size)
  {
    cubeSize = size;
    ...
  }

  public void start()
  {
    cubeX = 0;
    cubeY = -cubeSize;
    ...
  }
  ...
}
```

Fields

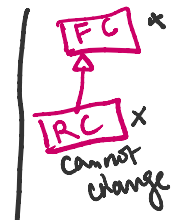
Constructor

Methods

The name of a constructor is always the same as the name of the class.

Name-relevant nouns

inheritance



Verbs

Fields

Fields

You name it!

private (or public) [static] [final]
datatype name;

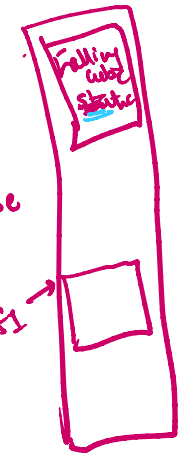
Usually private

May be present: means the field is shared by all objects in the class

May be present: means the field is a constant

int, double, etc., or an object: String, JButton, FallingCube, Timer

new FallingCube



Field qualifier

- Form of field declaration
  - **[qualifier] DataType fieldNames;**
    - qualifier : public, protected, private, static, final, volatile

```
int  anInteger, anotherIntegrer;
public String usage;
static long idNum = 0;
public static final double earthWeight = 5.97e24;
```

Field

- Access Modifier
  - Access Permission Level from other classes
  - public, protected, private
  - *Default/no declaration*: package

Group together a bunch of classes | Library classes
↳ directory
\Rang\math
Math.Cube2
only in the class.

Modifier	Class	Sub-Class	Same Package	All Class
private	O	X	X	X
package	O	X	O	X
protected	O	O	O	X
public	O	O	O	O

range

everywhere

Fields

- May have *primitive data types*:
  int, char, double, etc.

```
private int cubeX, cubeY;      // cube coordinates
...
private char randomLetter;     // cube letter
```

Fields

- May be objects of different types:

```
private FallingCube cube;
private Timer t;
private static final String letters;
```

↳ captital for classes

System.out.println( ).

Class not a class method.

Constructors

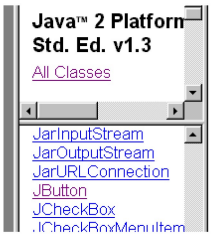
construct an object from a class

verb

↳ method

Constructors

- Constructors are like methods for creating objects of a class.
- Most constructors initialize the object's fields.
- Constructors may take parameters.
- A class may have several constructors that differ in the number or types of their parameters.
- All of a class's constructors have the same name as the class.



Constructors

Constructor Summary
JButton() Creates a button with no set text or icon.
JButton(Action a) Creates a button where properties are taken from the Action supplied.
JButton(Icon icon) Creates a button with an icon.
JButton(String text) Creates a button with text.
JButton(String text, Icon icon) Creates a button with initial text and an icon.

Classes
↓
Constructors needed

options

```
go = new JButton("Go");
```

Constructors (cont'd)

- Call them using the new operator:

cube = **new FallingCube**(CUBESIZE);

Calls FallingCube's constructor with CUBESIZE as the parameter

...

t = **new Timer**(delay, this)

Calls Timer's constructor with delay and this (i.e. this object) as the parameters (see Java docs for javax.swing.Timer)

new operator
↓
FallingCube( )
↓
Creates an object

Class constructors

- The purpose of the constructor is to initialize the instance variables
  - Default constructor || no parameter
  - Constructor overloading
    multiple

** forgetting to call a constructor

** constructor is invoked only when an object is first created. You cannot call the constructor to reset an object.

Constructors

- If a class has more than one constructor, they are "overloaded" and must have different numbers and/or types of arguments.
- Programmers often provide a "no-args" constructor that takes no arguments.
- If a programmer does not define any constructors, Java provides one default no-args constructor, which allocates memory and sets fields to the default values.

Unambiguously
→ choose a method

JButton
→ (String) 1 parameter types
→ (Icon)
(Action)
(String, Icon) 2

Constructors (cont'd)

```
public class Fraction
{
  private int num, denom;

  public Fraction ( )
  {
    num = 0;
    denom = 1;
  }

  public Fraction (int n)
  {
    num = n;
    denom = 1;
  }

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

  public Fraction (Fraction other)
  {
    num = other.num;
    denom = other.denom;
  }
  ...
}
```

"No-args" constructor

Copy constructor

Class
→ User choose

Reducing waste the number of lines of code
↳ DO NOT DUPLICATE CODE

Constructors

- Constructors of a class can call each other using the keyword this — a good way to avoid duplicating code:

```
public class Fraction
{
    public Fraction ()
    {
        this (0,1);
    }
    public Fraction (int n)
    {
        this (n, 1);
    }
    ...
    ...
    public Fraction (int p,
    int q)
    {
        num = p;
        denom = q;
        reduce ();
    }
    ...
```

1st line

1st line

instead

class.
int num

this( )
constructor

this. num

this

Operator new

- Constructors are invoked using new

```
Fraction  f1 = new Fraction ( );
Fraction  f2 = new Fraction (5);
Fraction  f3 = new Fraction (4, 6);
Fraction  f4 = new Fraction (f3);
```

0 / 1

5 / 1

2 / 3

2 / 3

Operator new (cont'd)

- You must create an object before you can use it; the new operator is a way to do it

Fraction f;

f is set to null

special object

null

f = new Fraction (2, 3);

Now f refers to a valid object

f = new Fraction (3, 4);

Now f refers to another object (the old object is "garbage-collected")

The this Keyword

this Keyword

this.varname

Invocation of Constructors

this(args);

```
class Point3D {
  double x;
  double y;
  double z;
  Point3D(double x, double y, double z)
{
  this.x = x;
  this.y = y;
  this.z = z;
  }
}
```

```
class ThisKeywordDemo {
  public static void main(String args[])
{ Point3D p1;
  Point3D p = new Point3D(1.1, 3.4, -2.8);
  System.out.println("p.x = " + p.x);
  System.out.println("p.y = " + p.y);
  System.out.println("p.z = " + p.z);
  } P2 = new Point3D(p1);
}
```

CompilerError.

HW

No parameters

1

2

Point3D

this

this.z = p1.z

class level variable

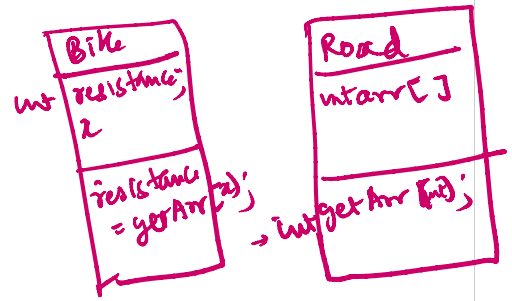
Method level variable

x =

= x

Calling one constructor from another

```
public class OOPClass
{
    int no_of_stud;
        public OOPClass( int i)
        {
                no_of_stud = i;
        }
        public OOPClass()
        {
                this(50);
        }
}
```



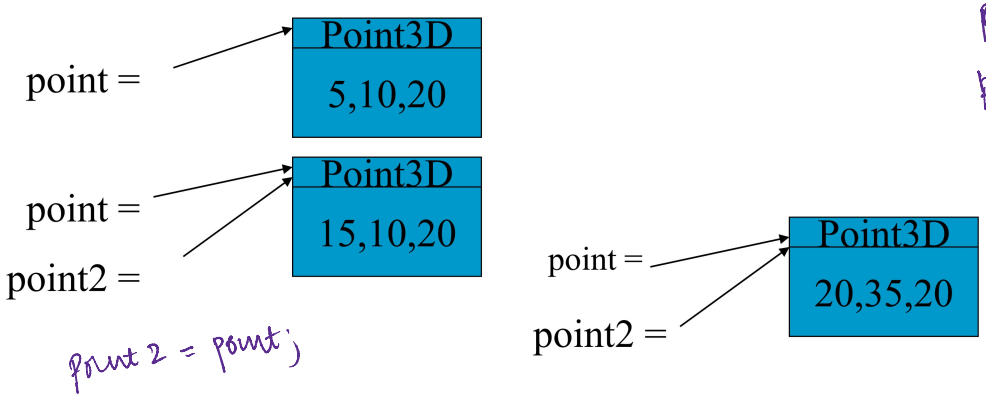
Minimise lines of code

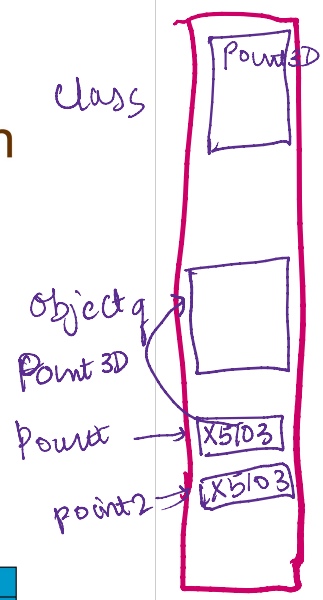
this(…) has to be the first line in the constructor

Object references

- A variable holds a memory location of an object.

point = new Point3D(,)



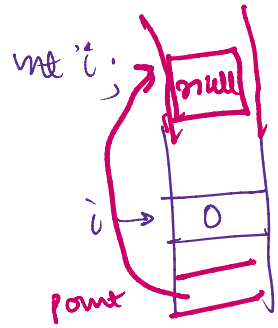


null references

- The null value is unique in that it can be assigned to a variable of any reference type whatsoever.
- Ex:
  - String middleInitial = null;
  - Car c = null;
- Compare objects
  - if (c == null)……

Point3D point;
↓
object pointer

int i;



{
}
else
{
}

Methods

Methods

- Call them for a particular object:

  cube.start();

But call *static* ("*class*") *methods* for the whole class, not a specific object:

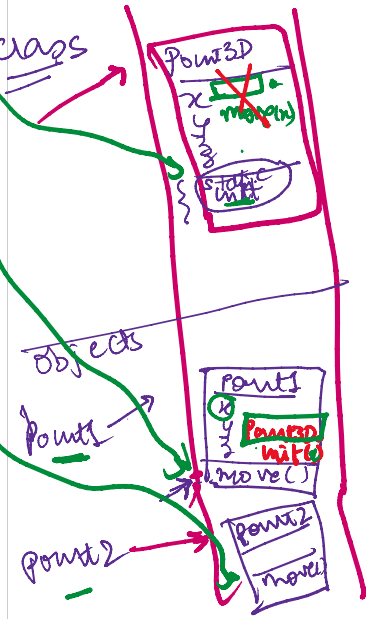
  y = **Math**.sqrt (x);

point1.init()

Point3D.init()

point1.move()
point2.move()

instance method

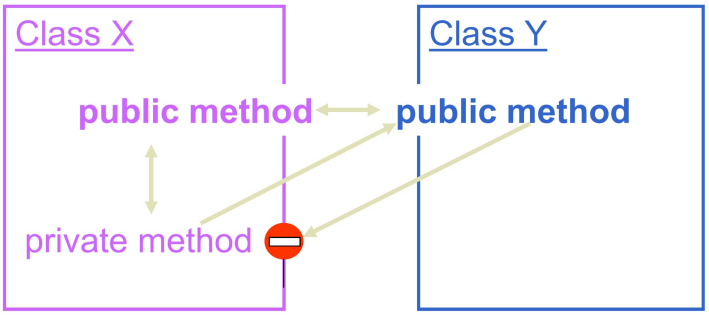


Methods

- Constructors and methods can call other public and private methods of the same class.
- Constructors and methods can call only public methods of another class.

Access specifier

public static void main()



Methods

```
public [or private]  returnType
                 methodName (type1 name1, ..., typeN nameN)
{
  ...
}
```

Body

Header

- To define a method:
  - decide between public and private (usually public)
  - give it a name
  - specify the types of arguments (formal parameters) and give them names
  - specify the method's return type or chose void
  - write the method's code

Hierarchy
package
lecture

Methods (cont'd)

- A method is always defined inside a class.
- A method returns a value of the specified type unless it is declared void; the return type can be any primitive data type or a class type.
- A method's arguments can be of any primitive data types or class types.

Empty parentheses indicate that a method takes no arguments.

```
public [or private]  returnType  methodName ( )
{   ...   }
```

Methods: Java Style

- Method names start with lowercase letters.
- Method names usually sound like verbs.
- The name of a method that returns the value of a field often starts with get:

  getWidth, getX

  The name of a method that sets the value of a field often starts with set:

  setLocation, setText

{ setX
move

x

Public getX
{
return x
}

setX

Method

- Form of Method Declaration

  [qualifier] returnType
  methodName(parameterList) {
  // method body
  }

  - **qualifier : modifier**, static, final, native, synchronized
  - **returnType : void unless return value**

```
class MethodExample {
    int simpleMethod() {
        //...
    }
    public void emptyMethod()  {  }
}
```

Method

- Method Qualifier
  - Access Modifier
    - Access Permission Level to Method from Other Class
    - Same as that of access modifier in field
  - **static**
    - static method, class method
    - Same role of Global function
    - Use only the static field of correspond class or the static method
    - Can be referred by only class name

```
ClassName.methodName;
```

Method

- **final**
  - Final method
  - Method which cannot be redefined in subclass
- **synchronized**
  - Synchronization method
  - Control the threads so that only one thread can always access the target
- **native**
  - To use the implementation written in other programming languages such as C language

Parameter

- Parameter Passing
  - Formal parameter
  - Actual parameter

value

```
void parameterPass(int i, Fraction f) {
    // ...
}
```

actual address . method change the object

- Local variable referred in method

```
class Fraction {
    int numerator, denominator;                    // Field
    public Fraction(int numerator, int denominator) {  // Parameter
        // ...
    }
}
```

{ numerator = }

this.numerator = numerator

{ method scope }

Parameter

- Call by value
- Call by reference
- main method

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

main()

- Pass in command line
  - public static void main(**String[ ] args**)

[command line]

		args[0]	**args[1]**	**args[2]**
java	ClassName	args1	args2	args3

Overloaded Methods

- Methods of the same class that have the same name but different numbers or types of arguments are called ***overloaded methods.***
- Use overloaded methods when they perform similar tasks:

```
public void move (int x, int y)   { ... }
public void move (double x, double y)
{ ... }
public void move (Point p)   { ... }

public Fraction add (int n)   { ... }
public Fraction add (Fraction other)   {
... }
```

Group 5

call dash.

define unambiguously

Overloaded Methods (cont'd)

- The compiler treats overloaded methods as completely different methods.
- The compiler knows which one to call based on the number and the types of the arguments:

```
public class Circle
{
  ...
   public void move (int x, int y)
   { ... }

   public void move (Point p)
   { ... }
  ...
}
```

```
Circle circle =  new Circle(5);

circle.move (50, 100);
...
Point center =
            new Point(50, 100);
circle.move (center);
...
```

Method Overloading

- Case of the same method name, but different in no. of parameter and type

```
void  methodOver(int i)  {  /* . . . */  }          // the first form
 void  methodOver(int i, int j){  /* . . . */ }// the second form
```

  – In case of method overloading, compilers do the following :
    - ✶Seek the method having the same parameter type
    - Seek the method having the parameter which can be converted by basic type casting

Method Overloading

```
public class  MethodOver {
        void someThing() {        // ...
       }
       void someThing(int i) {    // …
       }
       void someThing(int i, int j) {    // …
       }
       public static void main(String[] args) {
             MethodOver  m = new MethodOver();
             m.someThing();
             m.someThing(526);
             m.someThing(54, 526);
       }
}
```

Static

Static Fields

- A *static* field (a.k.a. *class field* or *class variable*) is shared by all objects of the class.
- A static field can hold a constant shared by all objects of the class:

```
public class RollingDie
{
  private static final double slowDown = 0.97;
  private static final double speedFactor = 0.04;
  ...
```

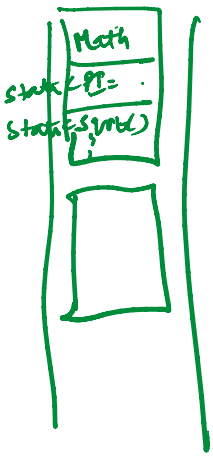
Reserved words:
static
final

- A non-static field (a.k.a. *instance field* or *instance variable*) belongs to an individual object.

Static Fields (cont'd)

- Static fields are stored with the class code, separately from non-static fields that describe an individual object.
- Public static fields, usually global constants, are referred to in other classes using "dot notation": ClassName.constName

```
double area = Math.PI * r * r;
setBackground(Color.blue);
c.add(btn, BorderLayout.NORTH);
System.out.println(area);
```



Static Fields (cont'd)

- Usually static fields are NOT initialized in constructors (they are initialized either in declarations or in public static methods).
- If a class has only static fields and does not have <u>any</u> non-static (instance) fields, there is no point in creating objects of that class (all of them would be identical).
- Math and System are examples of the above. In fact, they have no public constructors and cannot be *instantiated.*

Static Methods

- Static methods can access and manipulate a class's <u>static</u> fields.
- Static methods <u>cannot</u> access non-static fields or call non-static methods of the class.
- Static methods are called using "dot notation": ClassName . statMethod(...)

```
double x = Math.random();
double y = Math.sqrt (x);
System.exit();
```

Instance Methods

- Non-static methods are also called *instance methods.*
- An instance method is called for a particular object using “dot notation”:

  objName . instMethod(...);

- Instance methods can access ALL fields and call ALL methods of their class — both class and instance fields and methods.

Static (Class) vs. Non-Static (Instance)

```
public class MyClass
{
    public static final int statConst;
    private static int statVar;

    private int instVar;
    ...
    ...

    public static int statMethod(...)
    {
        statVar = statConst;
        statMethod2(...);

        instVar = ...;
        instMethod(...);
    }

    public int instMethod(...)
    {
        statVar = statConst;
        inst Var = statConst;
        instVar = statMethod(...);
        statVar = instMethod2(...);
        ...
    }

    public int instMethod2(...)
    {
        ...
    }
    ...
}
```

All OK

OK

Error!

class

main

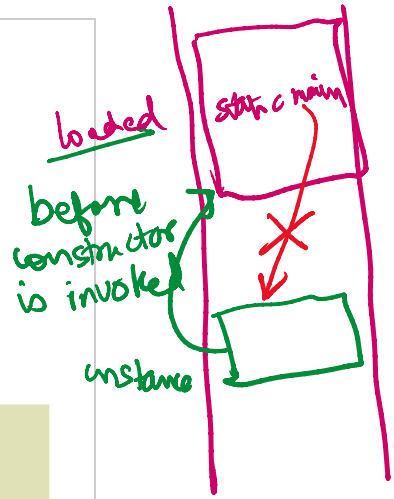
Static vs. Non-Static (cont'd)

- Note: main is static and therefore cannot access non-static fields or call non-static methods of its class:

```
public class Hello
{
  private String message = "Hello, World";

  public static void main (String[ ] args)
  {
     System.out.println (message);
  }
}
```

non static

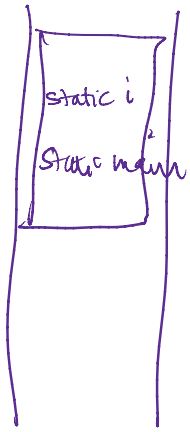
Error: non-static variable message is used in static context (main)



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> **}** also executed

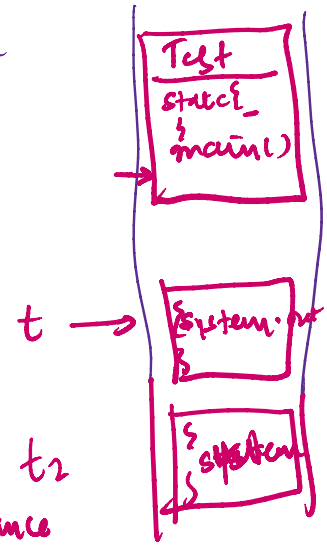


Static Initialization Statements

```
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();
    }
}
```

javac Test.java
java Test

Static once
Non-static block t every instance
Non-static block t2



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;
    //...
}
```

order

declaration
initialization
use

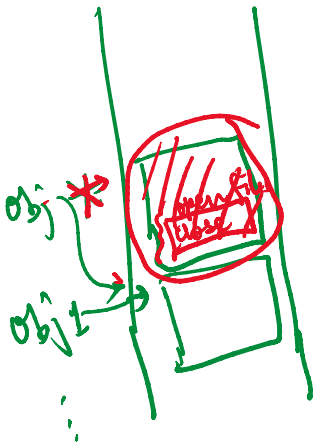
  - Executed earlier than constructor

static

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.

Code Others



Fraction Methods

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

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

```
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
  }
}
```

f1 = new Fraction ( )
f2 = new Fraction ( )

f1 → add
f2 → add

f1+f2

f1.add(f2);
f1+f2
f1=f1+f2

return type void

return f1

Fraction f3 = f1.invert(f2)

Assign Q6 → JavaDoc

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

Fraction f3; if (f3 == null) { }

object loaded

null

- 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:

f1.inverse()
num=0;

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

=0

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

good programmer

f2 = f1.inverse();
if (f2 == null) {
syst- - - - - - }

f2 is null
input of f1 is 0

user

f2.add(f1);

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);
...
```

value copied

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

Passing Objects as Arguments

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

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

address/pointer

f1: addr1

f2: addr2

copy

5/17

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

f: addr2

sum:

// f1.add (f2)
f1+f2

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

libraries
System.out
Math.PI
.sqrt(PI)
java.lang
default imported

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

bundled

AI

pkg

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.*;
```

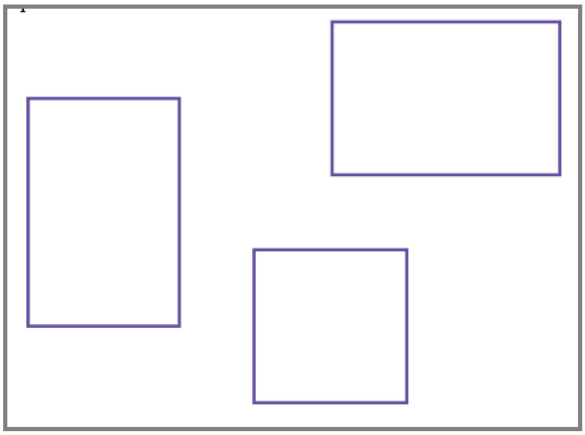
Imports all classes from awt, awt.event, and swing packages

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

Rectangular Shapes and Rectangle Objects

→ Rectangle
→ Use
Drawing Shapes

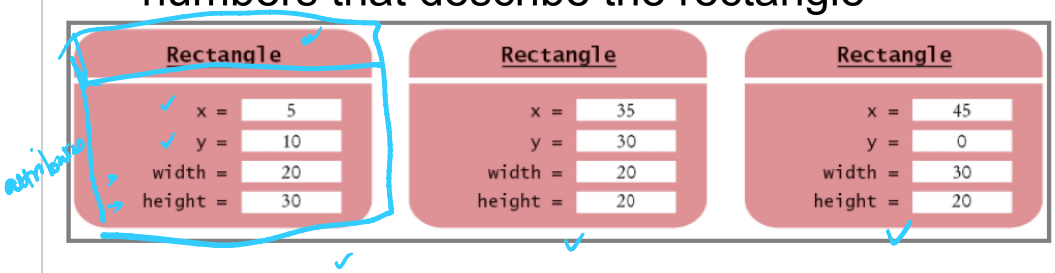
- Objects of type `Rectangle` *describe* rectangular shapes

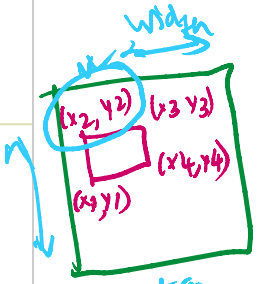


Rectangular Shapes

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





Abstraction
top left
(x,y)
width
Height.

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

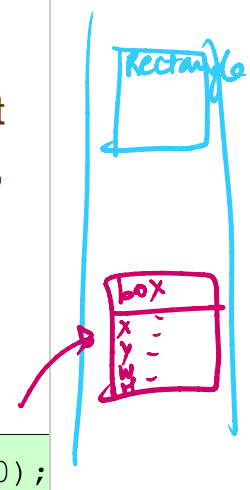
- `new Rectangle(5, 10, 20, 30)`

Top left

- 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);
```

constructor



Constructing Objects

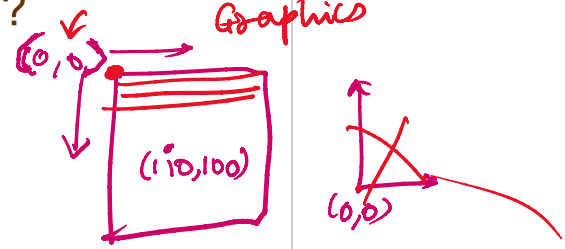
- The process of creating a new object is called *construction*
- The four values `5`, `10`, `20`, and `30` are called the *construction parameters*
- Some classes let you construct objects in multiple ways

```
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?

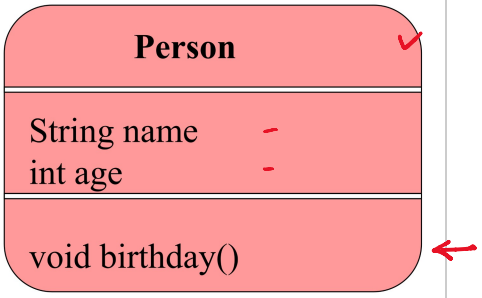
Computer Graphics



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);
   }
}
```

Q7

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

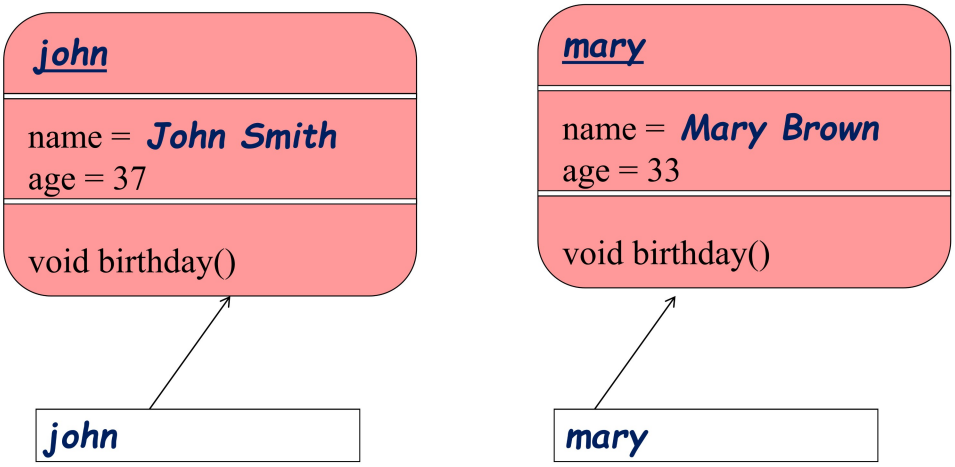
Person
String name int age
void birthday()

Creating and using objects

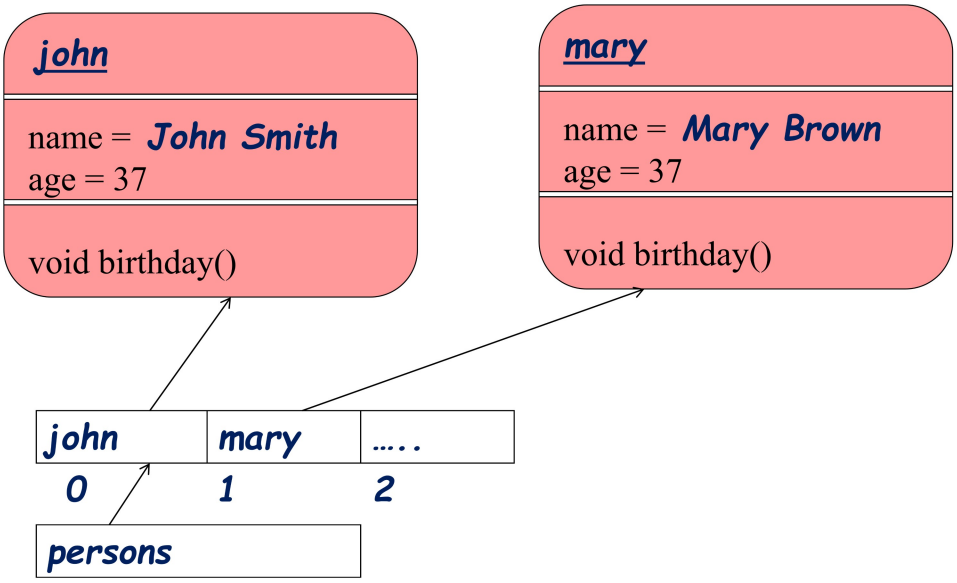
```
public class HelloWorldPerson {
   public static void main(String[] args){
      Person john;                // Declaring object
       john = new Person ( );  // creating object
       john.name = "John Smith";
       john.age = 37;

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

setup
↳ correct date
birthday() invoked.
5 people

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;
….
```