Java Classes and Objects (cont.)

A program with two classes

- As the source file grows it may be more convenient to implement each class in a separate file
- To compile both files together:

`javac StudentAccoun.java AccountSystem.java`

File: StudentAccount.java

```
class StudentAccount
{
    private String sName;
    public String getName() { return sName; }
    public void setName(String name) { sName = name; }
}
```

File: AccountSystem.java

```
import java.util.Scanner;

class AccountSystem
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner( System.in );
        System.out.print(" Enter a name: ");
        String aName = keyboard.next(); // read a word from the input buffer
        StudentAccount stud1 = new StudentAccount(); // create an object
        sdud1.setName( aName );  // call a public method to set private field sName
        String name1 = stud1.getName(); // get a private field via a public method
        System.out.println("Student's name: " + name1);
    }
}
```

Object Constructors

- This basic version of the application works well providing that setName() is called after an object is declared. Otherwise, the data field remains initialised with a default value (null or 0)
- To guarantee that data fields are always initialized, Java uses object **constructors**
- Every class in Java has at least one constructor
- If you do not declare a constructor in your class, Java will implicitly add a default constructor that will initialise fields to default values

```
StudentAccount stud1 = new StudentAccount();
```

- A default constructor is a method that has exactly the same name as the class.
  It doesn't take any parameters.

This is a call of the default constructor

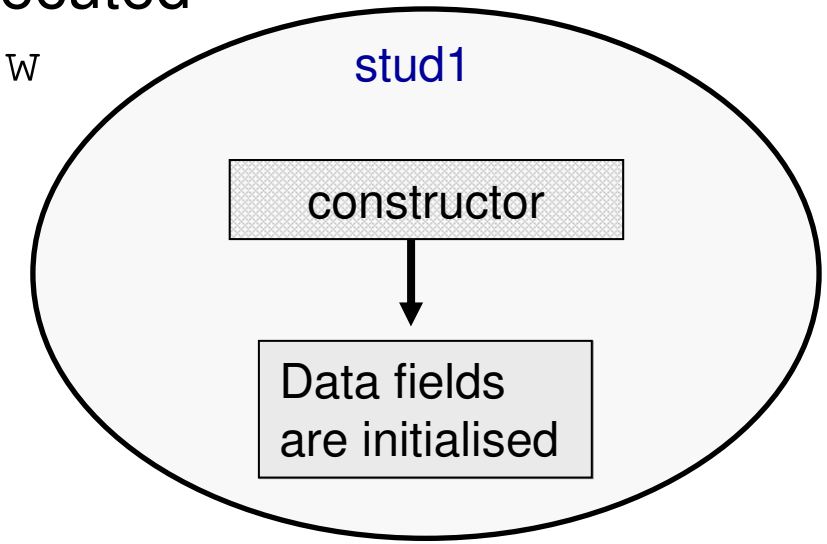
What are these brackets doing here? Is this a function call?

Constructors

- A constructor is <u>automatically</u> executed when an object is created by `new`

```
Account stud1 = new Account();
```

1. `new` allocates a memory block for an object
2. Default constructor Account() is called automatically to initialise fields



- Constructors have the following properties:
  - The name of the constructor is the same as the name of the class
  - A constructor does not return anything not even **`void`**
  - A class can have more than one constructor. In this case all constructors must have:
    1. the same name
    2. different sets of parameters

Constructors

File: StudentAccount.java

```
class StudentAccount    {
    private String sName;
    private String sEmail;
    private int    sNumber;

    /* default constructor */
    public StudentAccount() {
        sName = "";
        sEmail = "";
        sNumber = 0;
    }

    /* constructor with a parameter */
    public StudentAccount( String name ) {
        sName = name;
        sEmail = name + "@uowmail.edu.au";
        sNumber = (int)(1000.0 + 5000.0*Math.random());
    }
    public String getName() { return sName; }
    public void setName(String name) { sName = name; }
}
```

If you define a constructor with a parameter, a default constructor will not be generated by the compiler automatically

If you need a default constructor, define it explicitly together with other constructors

Declaration of Objects

File: AccountSystem.java

```
import java.util.Scanner;
import static java.lang.System.*;

class AccountSystem
{
    public static void main(String[] args)
    {
        .  .  .
        StudentAccount stud1 = new StudentAccount();

        .  .  .
        StudentAccount stud2 = new StudentAccount( name );

    }
}
```

Invoke a default constructor

Invoke a constructor with one parameter

Simple initialisation

- *Why is the initialization so complicated?*
- *Can I explicitly initialize fields with literals without defining a constructor?*

```
class Example
{
    private String name   = "Peter";
    private int number  = 10;
    private double temperature = 15.0;
    .   .   .
}
```

- You can use explicit initialisation values without defining a constructor. However, the compiler will still automatically generate a default constructor and place these initialisation values in it
- If you define a constructor with parameters, you may need to define a default constructor too, so initialization values will be ignored

Explicit initialization of values has a limited use

Simple initialisation

- *Does it mean that I should avoid explicit initialisation of fields?*

  Not always, it can be useful for certain fields

```
class Example
{
    private final double GRAVITY_ACCELERATION = 9.8;
    private static double payRate = 25.0;
    private String name;
    private int number;
    private double temperature;

    public Example () { // default constructor
        name = "";
        number = 1.0;
        temperature = 100.0;
    }
   .   .   .
}
```

- Explicit initialisation is commonly used for constants and static fields

Static initializers

- Static fields are initialised **only once** when the **class is first loaded** and then they are shared among all objects of this class
- Static fields are commonly known as **class variables**

- Non-static fields are initialised when a new object of the class is instantiated. They have no existence without instances (objects).
- Non-static fields are commonly known as **instance variables**

Quiz: Considering the following class definition

```
class HRData {
     public static double payRate = 25.0;
     public double hours;
}
.   .   .
public static void main(String[] args) {
   double rate = HRData.payRate;
    .    .    .
```

Why can you access payRate before any object of this class has been declared? Why is the class name (**`HRData.`**) specified to access it?

A program with two classes

File: StudentAccount.java

```
class StudentAccount    {
    private String sName;
    private String sEmail;
    private int    sNumber;
        .   .   .
    /* constructor with a parameter */
    public StudentAccount( String name ) {
        sName = name;
        sEmail = createEmail();
        sNumber = createSNumber();
    }

    private String createEmail() {
        return sName + "@uow.edu.au";
    }

    private int createSNumber()  {
        return (int)(1000.0 + 5000.0*Math.random());
    }
      .   .   .
}
```

To make constructors simpler and your code easier to follow, you can move all complex manipulations with data into separate methods

These methods are declared private because they are supplementary methods of this class and are supposed to be used only internally

Constructors can call methods, but methods can't call constructors

A program with two classes

File: AccountSystem.java

```
import java.util.Scanner;
import static java.lang.System.*;

class AccountSystem
{
    public static void main(String[] args)
    {
      Scanner keyboard = new Scanner( System.in );
      System.out.print(" Enter a name: ");
      String name = keyboard.next();
      StudentAccount stud1 = new StudentAccount( name );

      println( "Student's name: " + stud1.getName() );
      println( "Student's email: " + stud1.getEmail() );
      println( "Student's number: " + stid1.getNumber() );
    }
}
```

Invoke a constructor with one parameter

This functionality can be moved to a separate method printReport()

A program with two classes

File: AccountSystem.java

```
import java.util.Scanner;
import static java.lang.System.*;

class AccountSystem
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner( System.in );
        System.out.print(" Enter a name: ");
        String name = keyboard.next();

        StudentAccount stud1 = new StudentAccount( name );

        printReport( stud1 );

    }
    private static void printReport( StudentAccount acnt ) {
        println( "Student's name: " + acnt.getName() );
        println( "Student's email: " + acnt.getEmail() );
        println( "Student's number: " + acnt.getNumber() );
    }
}
```

A reference variable is passed to the method as a parameter

Two ways of passing parameters

There are two ways how parameters are passed to methods

1. Pass-by-Value

```
public int method1()
{
  int ap = 5;
  method2( ap );
  . . .
  println("value=" +  ap );
}
```

A copy of `ap` is created (rvalue)

5

The copy is assigned to the formal parameter `fp`

```
public void method2( int fp )
{
  fp *= 2;
  println("Value = " + fp );
}
```

Local variable ap is not affected by the operations in mehtod2() because only a copy of ap has been passed

All modifications of `fp` have only local effect

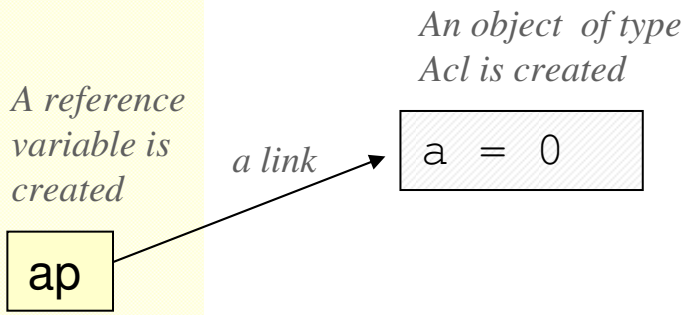
value = 5

Value = 10

Two ways of passing parameters

2. Pass-by-Reference

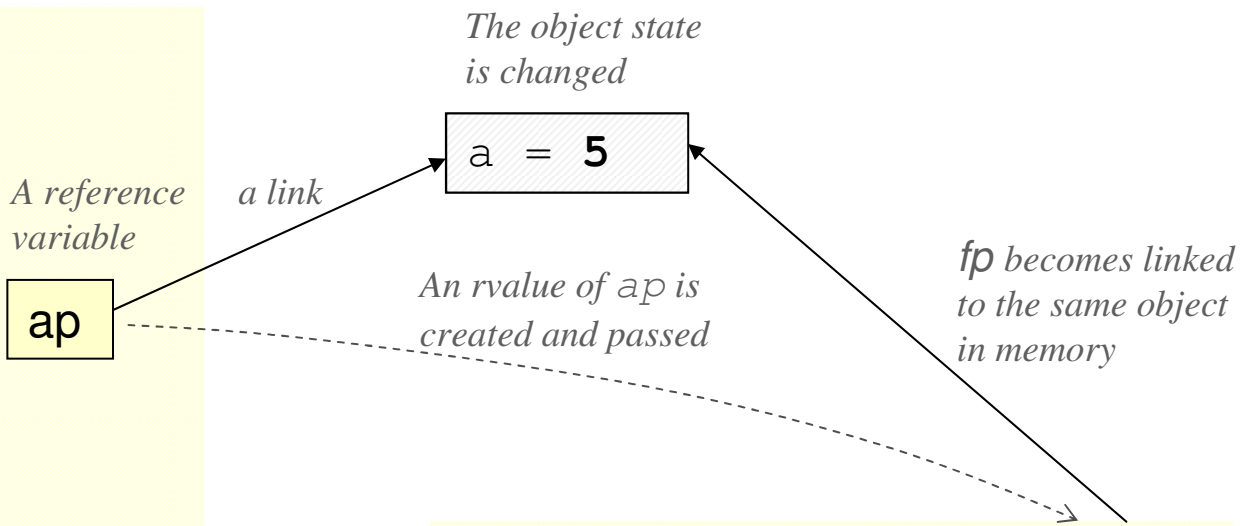
```
class Acl {
  public int a;
}
  . . .
public int method1()
{
  Acl ap;

  ap = new Acl();
  .   .   .

  .   .   .
}
```



Two ways of passing parameters

2. Pass-by-Reference

```
class Acl {
  public int a;
}
  . . .
public int method1()
{
  Acl ap;

  ap = new Acl();

  ap.a = 5;

  method2( ap );
  . . .
  . . .
}
```

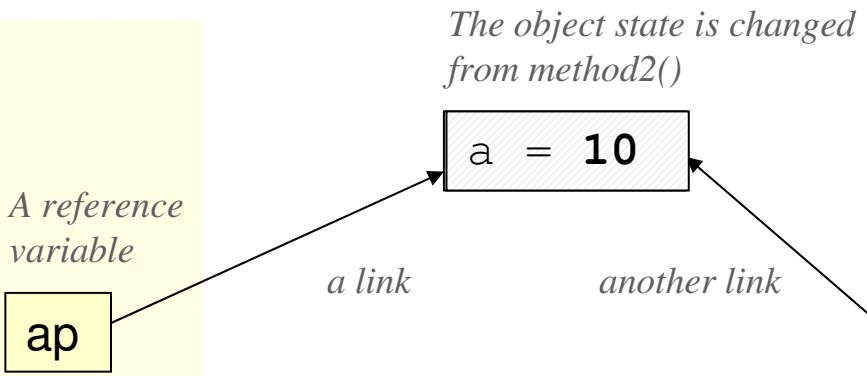



```
public void method2( Acl fp )
{
   .    .    .
}
```

Two ways of passing parameters

2. Pass-by-Reference

```
class Acl {
  public int a;
}
  . . .
public int method1()
{
  Acl ap;

  ap = new Acl();

  ap.a = 5;

  method2( ap );
  . . .
  println("value=" +  ap.a );
}
```




```
public void method2( Acl fp )
{
  fp.a = 10;
  println("Value=" +  fp.a );
}
```

value = 10

Value = 10

Passing by reference

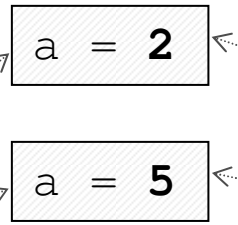
- Every time you pass a reference variable to a method, you actually pass a link to the object instead of the object itself

  OK, can I move all complex manipulations with objects into methods and all data manipulations done locally in those methods will have a global effect ?

- Manipulations with objects through their reference variables can be sometimes tricky. Detailed analysis and good understanding are essential !

```
. . .
public int method1()
{
 Co r1 = new Co(2);

 Co r2 = new Co(5);

 swap( r1, r2 );
  . . .

}
```

Swapping p1 and p2 has no affect on r1 and r2

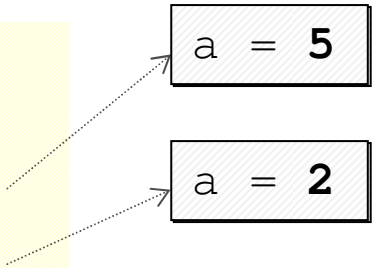



```
public void swap(Co p1, Co p2 )
{
  Co temp = p1;

  p1 = p2;
  p2 = temp;
}
```

Quiz

- How to swap objects ?

```
. . .
public int method1()
{
 Co r1 = new Co(2);
 Co r2 = new Co(5);
 swap( r1, r2 );
  . . .

}
```



`this` in Constructors

- As constructors are special methods defined in a class, can constructors call each other?
- When a class has multiple constructors, there is a simple way how one constructor can be called from another constructor

```
class Circle
{
    private double xCr, yCr, radius;   // fields

     /* - a constructor with three parameters */
    public Circle( double x, double y, double r ) {
       xCr = r; yCr = y; radius = r;
    }

    /* - a constructor with one parameter */
    public Circle( double r ) {
       this( 0.0, 0.0, r); // invokes a constructor with 3 parameters
    }
}
```

You can call constructors only from other constructors of the same class

Quiz

- If you define in a class constructor with three parameters, you have to define a default constructor too
- How to 'reuse' a constructor with three parameters when you define a default constructor?

```
class Circle
{
    private double xCr, yCr, radius;   // fields

     /* -- a constructor with three parameters --*/
    public Circle( double x, double y, double r ) {
        xCr = r;   yCr = y;   radius = r;
    }

     /* -- a default constructor --*/
    public Circle() {


    }   this( 0.0, 0.0, 1.0); // calls a constructor with 3 parameters
}
```

Copy Constructor

```
public int method()  {
  Account a1 = new Account("Saving");
  Account a2 = a1;
}
```

Object

- Assignment of reference variables only creates another link to an existing object

  How object clones can be created?

- To create clones of objects you need to define a copy constructor

```
class Circle
{
    private double xCr, yCr, radius;   // fields

   /*- a copy constructor, it takes only one parameter -*/
    public Circle( Circle source ) {
        this( source.xCr, source.yCr, source.radius );
    }
     .   .   .
}
```

Get fields from the source and invoke a constructor

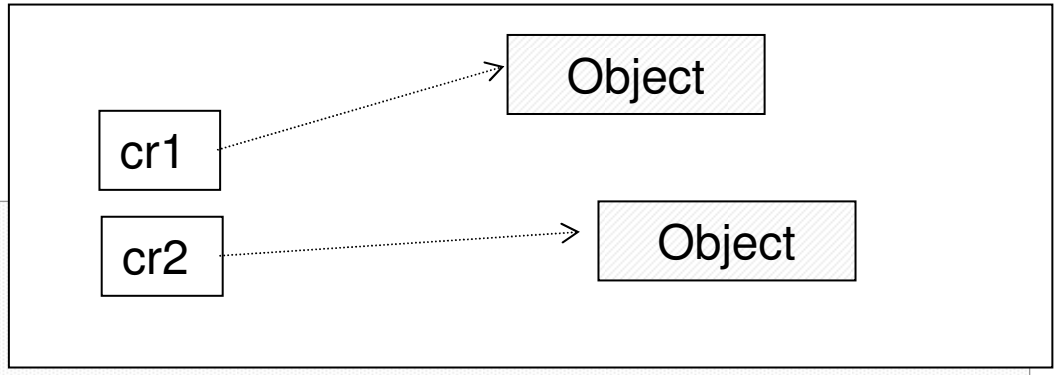
Copy Constructor

- If you do not want to use **`this`** in a copy constructor, you can assign fields explicitly without invoking other constructors

```
class Circle
{
    private double xCr, yCr, radius;   // fields

   /*- a copy constructor, it takes only one parameter
       and this parameter must be of type class_name  -*/
    public Circle( Circle source ) {
        xCr = source.xCr;
        yCr = source.yCr;
        radius = source.radius;
    }
     .   .   .
}
```

If you do not define a copy constructor, the compiler **will not** do it for you

Copy Constructor

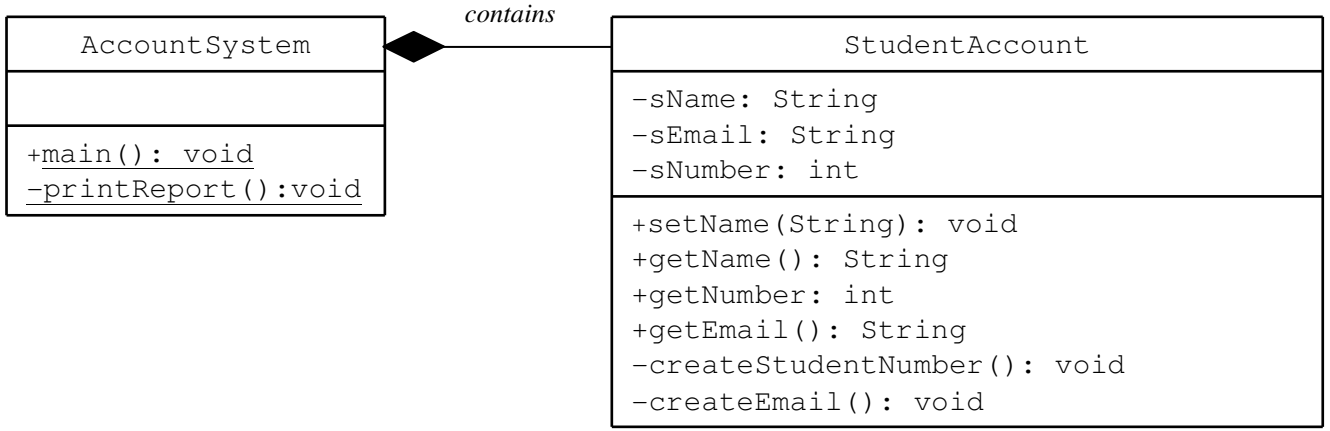



```
class Circle  {
.   .   .
}

class Example
{
    public static void main(String[] args)
    {
        /* -- Create a new object -- */
        Circle cr1 = new Circle( 5.5, 4.5, 3.0 );

        /*-- Create a clone -- */
        Circle cr2 = new Circle( cr1 );

        cr2.setRadius( 10.0 );                  // doesn't affect cr1
        cr1.setparameters( -7.0, -3.5, 2.0 );   // doesn't affect cr2
    }
}
```

A source object is provided as the only actual parameter

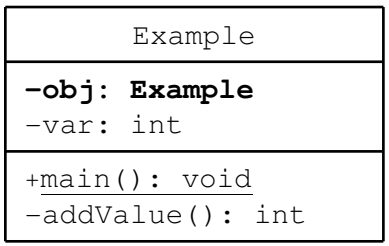
Solution 2: main() declares and object

- Although definition of another class and composition is a commonly used approach that can resolve limitations of the static main(), there is an alternative solution that reflects flexibility of OOD



Solution 2:

Define a class that instantiates an object of its own type



- *Although main() is a static method, the class contains non-static fields and methods*
- *One of the fields is of type Example*
- *How can they be accessed from the static main()?*

Solution 2: main() declares and object

```
class Example
{
    private int fld = 0;            // a non-static field
    private static int sfld = 0;   // a static field

    public static void main(String[] args)
    {
        Example mex = new Example(); // declare an object

        sfld = 2; // you can access static fields directly
        mex.fld = 5; // you need to reference the object
        sfld = mex.addTwo( mex.fld );
        System.out.println( sfld );
    }

    public int addTwo( int a ) // a non-static method
    {
       return a + 2;
    }
}
```

Quiz

- You **cannot** use **`new`** this way although it can be compiled. Why?

```
class Example
{
    private int fld = 0;
    static Example ex1;  // OK
    Example mex = new Example(); // <- NO

    public static void main(String[] args)  {
        ex1 = new Example();  // OK
        mex.fld = 5;
        int result = mex.addTwo( mex.fld );
        System.out.println( result );
    }

    public int addTwo( int a ) // a non-static method
    {
       return a + 2;
    }
}
```

Relationship between classes

Composition

File: AccountSystem.java

```
class AccountSystem
{
    private StaffAccount staffAct;
    private StudentAccount studAct;

    public AccountSystem(String name)
    {
        staffAct = new StaffAccount( name );
        studAct = new StudentAccount( name );
        .   .   .

    }
}
```

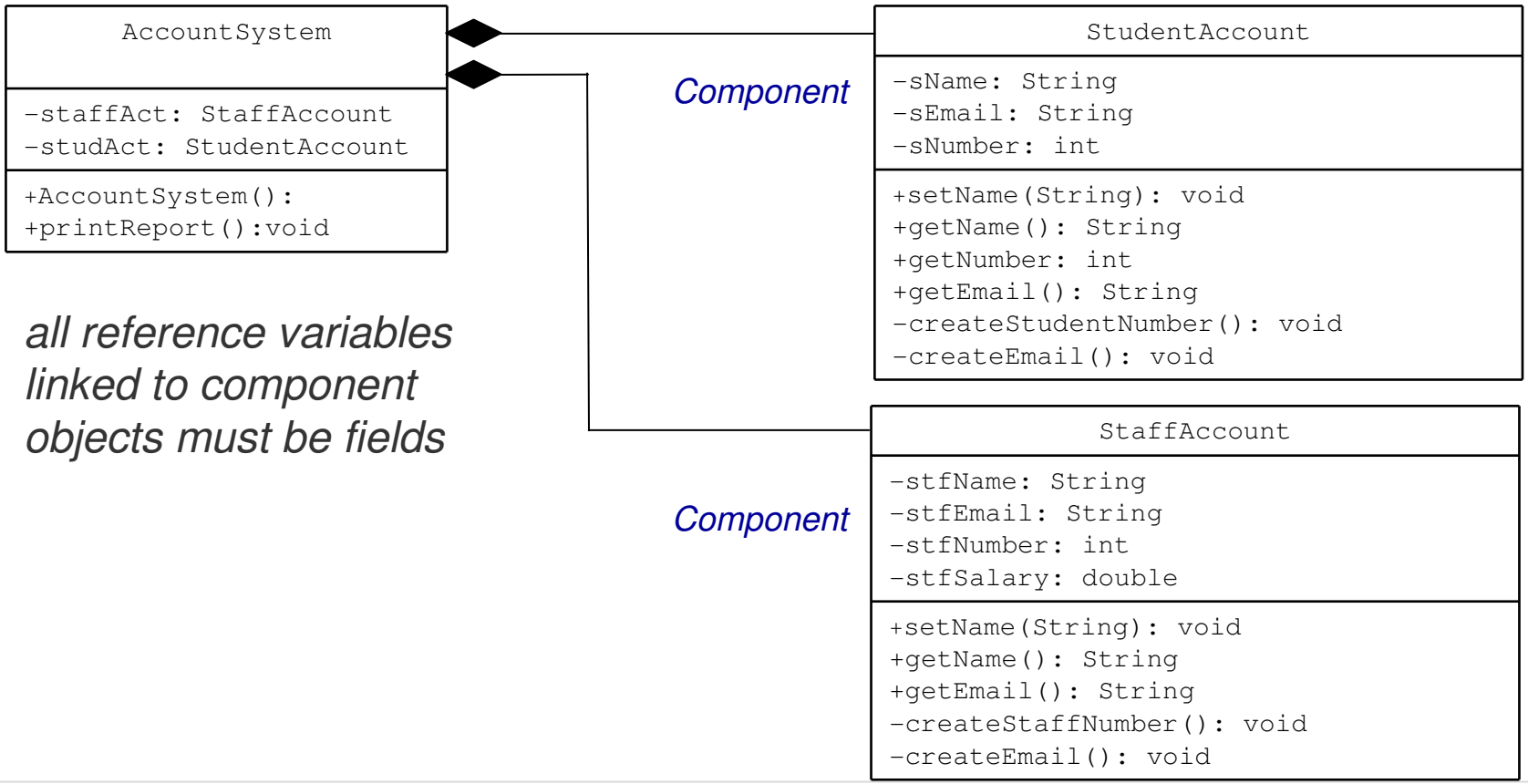
- Objects of related classes (such as **`staffAct and studAct`**) are instantiated by **`AccountSystem`** constructor

- If an object of **`AccountSystem`** class is destroyed, **`stuffAct and studAct`** will cease to exist too. Component objects have no existence outside of the container - one of the major properties of composition

Composition

Relationship between classes

A container class

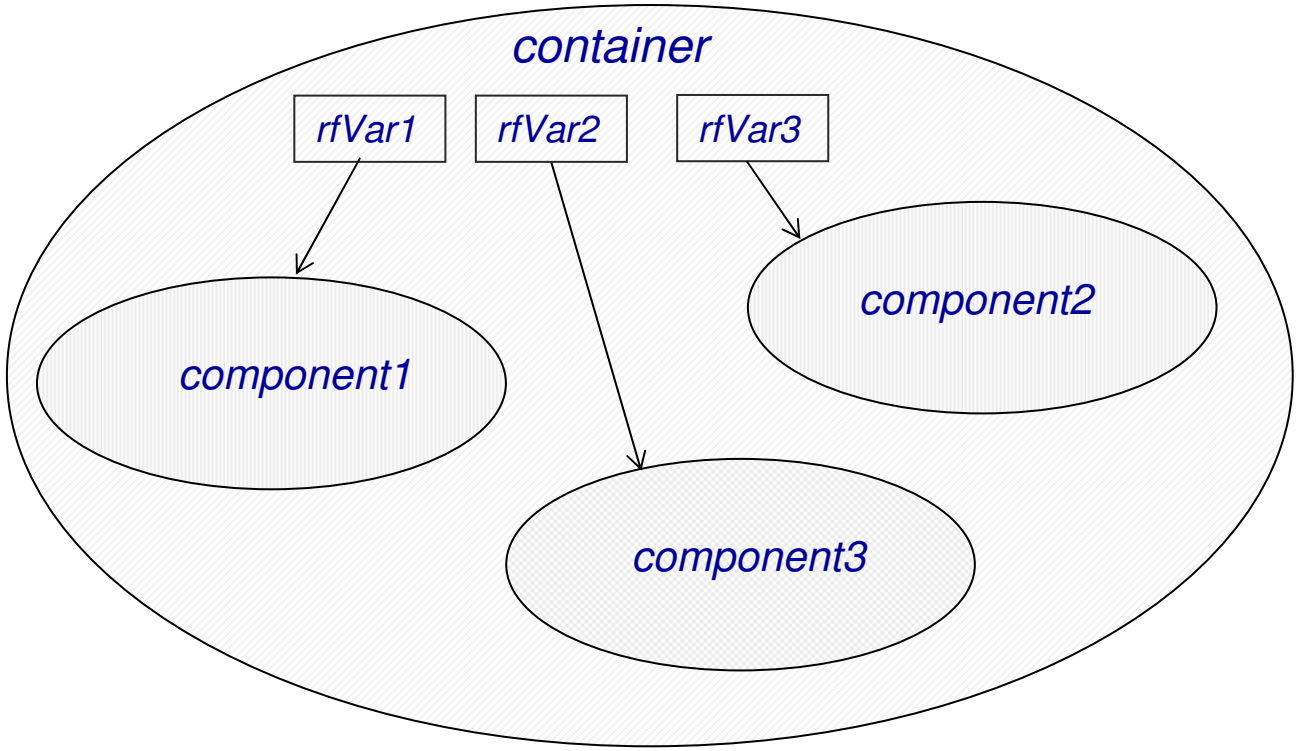


all reference variables linked to component objects must be fields

Composition

Relationship between objects

Animated objects may not be visible on printed slides

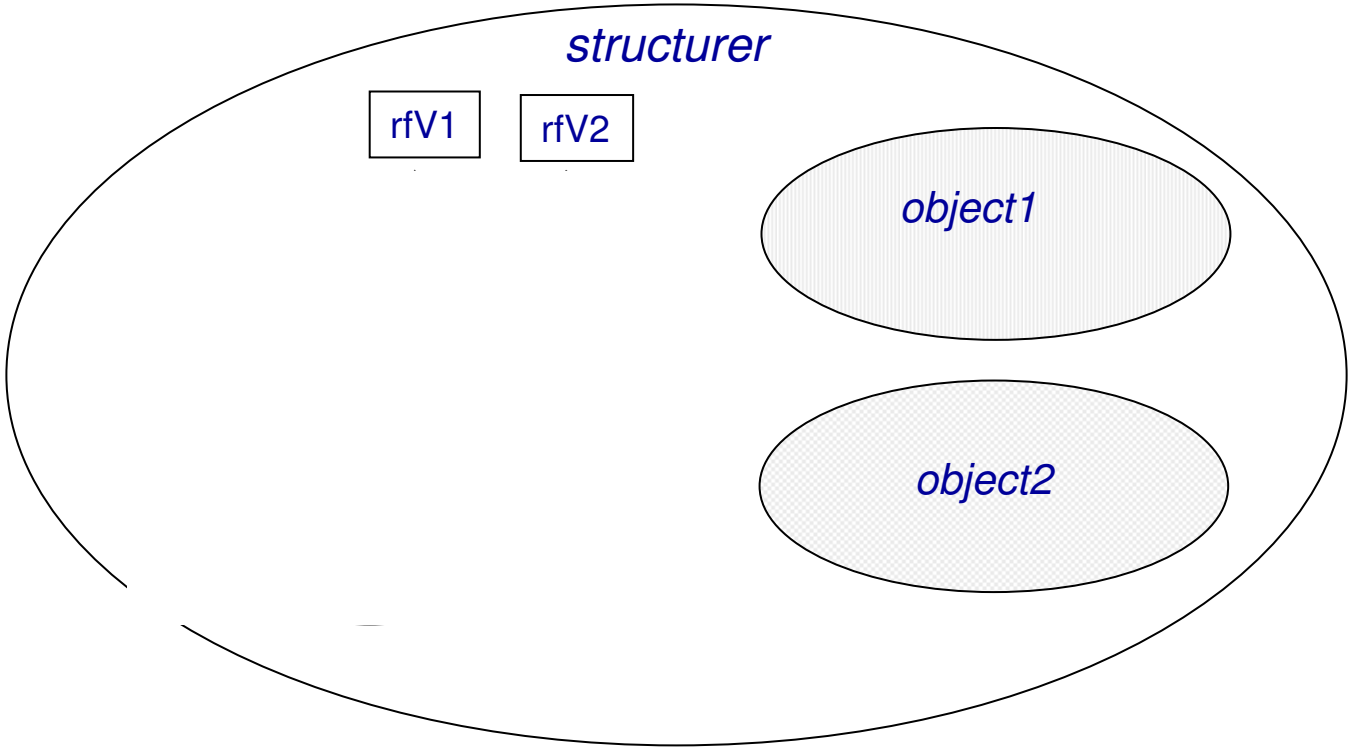


- Component objects are created by the container itself using **`new`** operator
- All components are destroyed when the container is destroyed

Association

This relationship is more complex to implement

Animated objects may not be visible on printed slides



- Component objects are instantiated by another object called the structurer
- *object3* does not create *object1* and *object2* it is only given links to them through passing reference variables (dependency injection)
- When *object3* is deleted, *object1* and *object2* are not affected

Class design hints

- Declare data fields private
  - Doing anything else violates encapsulation – information hiding
- Always initialize data fields
  - Don't rely on the default values
  - Implement all relevant constructors including a copy constructor
- Not all private fields need corresponding public methods to access them. Some fields may only be used internally and should never be access from outside
- Don't use too many basic types in a class, use composition

```
class Student {
  private String name;
  private int stNumber;
  private String street;
  private String city;
  private String state;
  private int postcode;
}
```

```
class Student {
  private String name;
  private int stNumber;
  private Address adr;
  …
}
```

```
class Address {
  private String street;
  private String city;
  private String state;
  private int postcode;
  …
}
```

Java API – class library

http://docs.oracle.com/javase/8/docs/api/

Java™ Platform Standard Ed. 8

All Classes All Profiles

Packages

java.applet
java.awt

MarshalException
MarshalledObject
Marshaller
Marshaller.Listener
MaskFormatter
Matcher
MatchResult
Math
MathContext
MatteBorder
MBeanAttributeInfo
MBeanConstructorInfo
MBeanException
MBeanFeatureInfo
MBeanInfo
MBeanNotificationInfo
MBeanOperationInfo
MBeanParameterInfo
MBeanPermission
MBeanRegistration

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

Java™ Platform Standard Ed. 8

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3
java.lang

Class Math

java.lang.Object
    java.lang.Math

```
public final class Math
extends Object
```

The class `Math` contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

Unlike some of the numeric methods of class `StrictMath`, all implementations of the equivalent functions of class `Math` are not defined to return the bit-for-bit same results. This relaxation permits better-performing implementations where strict reproducibility is not required.

By default many of the `Math` methods simply call the equivalent method in `StrictMath` for

From now on, you should be able to find methods you need from Java API
Familiarising yourself with basic Java API is part of learning Java programming

Suggested reading

Java: How to Program (Early Objects), 10th Edition

- Chapter 3: Introduction to Classes