

Object Oriented Programming using Java 3

Arrays & Classes

Arrays

For-each Loop

Command - Line Arguments

Procedural Programming

Object & Class

Class Outline

Methods

Constructor

Garbage Collection

this keyword

Method Overloading

Constructor Overloading

Passing Object to
Constructor

Argument Passing

Recursion

Static Members

Array of Objects

Array as Argument

Returning array from
Method

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Arrays

- Arrays are:
 - declared
 - created
 - initialized
 - used
- **Array Declaration**
 - declaring an array identifier
 - declaring the number of dimensions
 - declaring the data type of the array elements

type array-variable[];

or

type [] array-variable;

Arrays

[For-each Loop](#)[Command - Line Arguments](#)[Procedural Programming](#)[Object & Class](#)[Class Outline](#)[Methods](#)[Constructor](#)[Garbage Collection](#)[this keyword](#)[Method Overloading](#)[Constructor Overloading](#)[Passing Object to Constructor](#)[Argument Passing](#)[Recursion](#)[Static Members](#)[Array of Objects](#)[Array as Argument](#)[Returning array from Method](#)

Arrays...

- **Array Creation**

- After declaration, no array actually exists
- In order to create an array, we use the **new** operator:

```
type array-variable[];  
array-variable = new type[size];
```

- We can refer to the elements of this array through their indexes:

```
array-variable[index]
```

- The array index always starts with zero!

- **Array Initialization**

- Arrays can be initialized when they are declared:

```
int monthDays[] = {31,28,31,30,31,30,31,31,30,31,30,31};
```

Arrays

[For-each Loop](#)[Command - Line Arguments](#)[Procedural Programming](#)[Object & Class](#)[Class Outline](#)[Methods](#)[Constructor](#)[Garbage Collection](#)[this keyword](#)[Method Overloading](#)[Constructor Overloading](#)[Passing Object to Constructor](#)[Argument Passing](#)[Recursion](#)[Static Members](#)[Array of Objects](#)[Array as Argument](#)[Returning array from Method](#)

Multidimensional Arrays

- **Multidimensional arrays are arrays of arrays:**

- declaration:

```
int array[][];
```

- creation:

```
int array = new int[2][3];
```

- initialization:

```
int array[] [] = { {1, 2, 3}, {4, 5, 6} };
```

Arrays

[For-each Loop](#)[Command - Line Arguments](#)[Procedural Programming](#)

Object & Class

[Class Outline](#)[Methods](#)[Constructor](#)[Garbage Collection](#)[this keyword](#)[Method Overloading](#)[Constructor Overloading](#)[Passing Object to Constructor](#)[Argument Passing](#)[Recursion](#)[Static Members](#)

Array of Objects

Array as Argument

[Returning array from Method](#)

For-each Loop

For-each Loop

It is mainly used to traverse array or collection elements. It eliminates the possibility of bugs and makes the code more readable

```
public class TestArray {  
    public static void main(String[] args) {  
        int []arr = {11,22,33,44,55};  
        for (int i: arr) {  
            System.out.println(i);  
        }  
    }  
}
```

Arrays

For-each Loop

Command - Line Arguments

Procedural Programming

Object & Class

Class Outline

Methods

Constructor

Garbage Collection

this keyword

Method Overloading

Constructor Overloading

Passing Object to Constructor

Argument Passing

Recursion

Static Members

Array of Objects

Array as Argument

Returning array from Method

Command - Line Arguments

Command - Line Arguments

- A command-line argument is the information that directly follows the program's name on the command line when it is executed
- The command-line arguments are stored as strings in the **String** array passed to `main()`

```
class CommandLine {  
    public static void main(String args[]) {  
        for(int i=0; i<args.length; i++)  
            System.out.println("args[" + i + "]: " + args[i]);  
    }  
}
```

- `javac CommandLine.java`
- `java CommandLine this is a test`
- *All command-line arguments are passed as strings*

Procedural Programming

- data → nouns versus operations → verbs
- Procedural programming is verb- oriented:
 - decomposition around operations
 - operation are divided into smaller operations
- Drawbacks:
 - data is given a second- class status when compared with operations
 - difficult to relate to the real world: no functions in real world
 - difficult to create new data types: reduces extensibility
 - programs are difficult to debug: little restriction to data access
 - programs are hard to understand: many variables have global scope
 - programs are hard to reuse: data/functions are mutually dependent
 - little support for developing and comprehending really large programs
 - top- down development approach tends to produce monolithic programs

Arrays

For-each Loop

Command - Line Arguments

Procedural Programming

Object & Class

Class Outline

Methods

Constructor

Garbage Collection

this keyword

Method Overloading

Constructor Overloading

Passing Object to Constructor

Argument Passing

Recursion

Static Members

Array of Objects

Array as Argument

Returning array from Method

Object & Class

- Real world objects are things that have:
 - state
 - behavior
- Example: your dog
 - state: name, color, breed
 - behavior: sitting, barking, wagging tail, running
- A software **object** is a bundle of variables (state) and methods (operations)
- A **class** is a blueprint that defines the variables and methods common to all objects of a certain kind
 - Example: 'your dog' is a object of the class Dog
- An object holds values for the variables defined in the class
- A class is a **template** for an object, whereas an object is called an **instance** of the class

[Arrays](#)

[For-each Loop](#)

[Command - Line Arguments](#)

[Procedural Programming](#)

[Object & Class](#)

[Class Outline](#)[Methods](#)[Constructor](#)[Garbage Collection](#)[this keyword](#)[Method Overloading](#)[Constructor Overloading](#)[Passing Object to Constructor](#)[Argument Passing](#)[Recursion](#)[Static Members](#)

[Array of Objects](#)

[Array as Argument](#)

[Returning array from Method](#)

Class Outline

- *A basis for the Java language*
- Each concept we wish to describe in Java must be included inside a class
- A class defines a new data type, whose values are objects
- **Class Definition**
 - A class contains a **name**, several **variable** declarations (instance variables) and several **method** declarations. All are called **members** of the class

```
class classname {  
    type instance-variable-1;  
    ...  
    type instance-variable-n;  
    type method-name-1(parameter-list) { ... }  
    ...  
    type method-name-m(parameter-list) { ... }  
}
```

Arrays

For-each Loop

Command - Line Arguments

Procedural Programming

Object & Class

Class Outline

Methods

Constructor

Garbage Collection

this keyword

Method Overloading

Constructor Overloading

Passing Object to
Constructor

Argument Passing

Recursion

Static Members

Array of Objects

Array as Argument

Returning array from
Method

Class Outline...

- Example: Box class definition and usage

```
class Box {  
    double width;  
    double height;  
    double depth;  
}  
  
class BoxDemo {  
    public static void main(String args[]) {  
        Box mybox = new Box();  
        double vol;  
        mybox.width = 10;  
        mybox.height = 20;  
        mybox.depth = 15;  
        vol = mybox.width * mybox.height * mybox.depth;  
        System.out.println("Volume is " + vol);  
    }  
}
```

- **Compilation and execution**

- > javac BoxDemo.java
- > java BoxDemo

Arrays

For-each Loop

Command - Line Arguments

Procedural Programming

Object & Class

Class Outline

Methods

Constructor

Garbage Collection

this keyword

Method Overloading

Constructor Overloading

Passing Object to Constructor

Argument Passing

Recursion

Static Members

Array of Objects

Array as Argument

Returning array from Method

Class Outline...

- **Declaring Objects**

- *Everything in Java is an object*
- Declare a variable of the class type:

```
Box myBox;  
myBox = new Box();
```

- Allocates memory for a Box object and returns its address:

```
Box myBox = new Box();
```

- **Assigning Reference Variables**

- *Assignment copies address, not the actual value*

```
Box b1 = new Box();  
Box b2 = b1;
```

- Both variables point to the same object

[Arrays](#)[For-each Loop](#)[Command - Line Arguments](#)[Procedural Programming](#)[Object & Class](#)[Class Outline](#)[Methods](#)[Constructor](#)[Garbage Collection](#)[this keyword](#)[Method Overloading](#)[Constructor Overloading](#)[Passing Object to Constructor](#)[Argument Passing](#)[Recursion](#)[Static Members](#)[Array of Objects](#)[Array as Argument](#)[Returning array from Method](#)[Returning array from Method](#)

Methods

- General form of a method definition:

```
type name(parameter-list) {  
    ... return value; ...  
}
```

- Classes declare methods to hide their internal data structures, as well as for their own internal use
- Within a class, we can refer directly to its member variables:

```
class Box {  
    double width, height, depth;  
    void volume() {  
        System.out.print("Volume is ");  
        System.out.println(width * height * depth);  
    }  
}
```

Arrays

[For-each Loop](#)

Command - Line Arguments

Procedural Programming

Object & Class

[Class Outline](#)

Methods

[Constructor](#)[Garbage Collection](#)[this keyword](#)[Method Overloading](#)[Constructor Overloading](#)[Passing Object to Constructor](#)[Argument Passing](#)[Recursion](#)[Static Members](#)

Array of Objects

Array as Argument

[Returning array from Method](#)

Parameterized Method

- Parameters increase generality and applicability of a method:
 - Method without parameters:
*int square() { return 10*10; }*
 - Method with parameters:
*int square(int i) { return i*i; }*

```
void setDim(double w, double h, double d) {  
    width = w;  
    height = h;  
    depth = d;  
}
```

[Arrays](#)[For-each Loop](#)[Command - Line Arguments](#)[Procedural Programming](#)[Object & Class](#)[Class Outline](#)[Methods](#)[Constructor](#)[Garbage Collection](#)[this keyword](#)[Method Overloading](#)[Constructor Overloading](#)[Passing Object to Constructor](#)[Argument Passing](#)[Recursion](#)[Static Members](#)[Array of Objects](#)[Array as Argument](#)[Returning array from Method](#)

Constructor

- A constructor initializes the instance variables of an object
- It is called immediately after the object is created but before the **new** operator completes
 - it is syntactically similar to a method
 - it has the same name as the name of its class
 - it is written without return type; the default return type of a class constructor is the same class
- When the class has no constructor, the **default constructor** automatically initializes all its instance variables with zero

[Arrays](#)[For-each Loop](#)[Command - Line Arguments](#)[Procedural Programming](#)[Object & Class](#)[Class Outline](#)[Methods](#)[Constructor](#)[Garbage Collection](#)[this keyword](#)[Method Overloading](#)[Constructor Overloading](#)[Passing Object to Constructor](#)[Argument Passing](#)[Recursion](#)[Static Members](#)[Array of Objects](#)[Array as Argument](#)[Returning array from Method](#)

Constructor...

```
class Box {  
    double width;  
    double height;  
    double depth;  
    Box() {  
        System.out.println("Constructing Box");  
        width = 10; height = 10; depth = 10;  
    }  
    double volume() {  
        return width * height * depth;  
    }  
}
```

Arrays

For-each Loop

Command - Line
Arguments

Procedural
Programming

Object & Class

Class Outline

Methods

Constructor

Garbage Collection

this keyword

Method Overloading

Constructor Overloading

Passing Object to
Constructor

Argument Passing

Recursion

Static Members

Array of Objects

Array as Argument

Returning array from
Method

Constructor...

- **Parameterized Constructor**

```
Box(double w, double h, double d) {  
    width = w; height = h; depth = d;  
}
```

- **finalize()** Method:

- finalize method is invoked just before the object is destroyed
- implemented inside a class as:

```
protected void finalize() { ... }
```

- implemented when the usual way of removing objects from memory is insufficient, and some special actions has to be carried out

[Arrays](#)[For-each Loop](#)[Command - Line Arguments](#)[Procedural Programming](#)[Object & Class](#)[Class Outline](#)[Methods](#)[Constructor](#)[Garbage Collection](#)[this keyword](#)[Method Overloading](#)[Constructor Overloading](#)[Passing Object to Constructor](#)[Argument Passing](#)[Recursion](#)[Static Members](#)[Array of Objects](#)[Array as Argument](#)[Returning array from Method](#)

Garbage Collection

- Garbage collection is a mechanism to remove objects from memory when they are no longer needed
- Garbage collection is carried out by the garbage collector:
 - The garbage collector keeps track of how many references an object has
 - It removes an object from memory when it has no longer any references
 - Thereafter, the memory occupied by the object can be allocated again
 - Garbage collector is parts of the Java Run-Time Environment
 - The garbage collector invokes the **finalize()** method

[Arrays](#)[For-each Loop](#)[Command - Line Arguments](#)[Procedural Programming](#)[Object & Class](#)[Class Outline](#)[Methods](#)[Constructor](#)[Garbage Collection](#)[this keyword](#)[Method Overloading](#)[Constructor Overloading](#)[Passing Object to Constructor](#)[Argument Passing](#)[Recursion](#)[Static Members](#)[Array of Objects](#)[Array as Argument](#)[Returning array from Method](#)

this keyword

this keyword

- Keyword **this** allows a method to refer to the object that invoked it
- It can be used inside any method to refer to the current object

```
Box(double w, double h, double d) {  
    this.width = w;  
    this.height = h;  
    this.depth = d;  
}
```

```
Box(double width, double height, double depth) {  
    this.width = width;  
    this.height = height;  
    this.depth = depth;  
}
```

Arrays

For-each Loop

Command - Line Arguments

Procedural Programming

Object & Class

Class Outline

Methods

Constructor

Garbage Collection

this keyword

Method Overloading

Constructor Overloading

Passing Object to Constructor

Argument Passing

Recursion

Static Members

Array of Objects

Array as Argument

Returning array from Method

Method Overloading

```
class OverloadDemo {  
    void test() { System.out.println("No parameters"); }  
  
    void test(int a) { System.out.println("a: " + a); }  
  
    void test(int a, int b) { System.out.println("a and b: " + a + " " + b); }  
  
    double test(double a) { System.out.println("double a: " + a);  
                           return a*a;  
    }  
}
```

- When an overloaded method is called, Java looks for a match between the arguments used to call the method and the method's parameters
- When no exact match can be found, Java's automatic type conversion can aid overload resolution

[Arrays](#)

[For-each Loop](#)

[Command - Line Arguments](#)

[Procedural Programming](#)

[Object & Class](#)

[Class Outline](#)[Methods](#)[Constructor](#)[Garbage Collection](#)[this keyword](#)

[Method Overloading](#)

[Constructor Overloading](#)[Passing Object to Constructor](#)[Argument Passing](#)[Recursion](#)[Static Members](#)

[Array of Objects](#)

[Array as Argument](#)

[Returning array from Method](#)

Constructor Overloading

```
class Box {  
    double width, height, depth;  
    Box(double w, double h, double d) {  
        width = w; height = h; depth = d;  
    }  
    Box() {  
        width = -1; height = -1; depth = -1;  
    }  
  
    Box(double len) {  
        width = height = depth = len;  
    }  
    double volume() { return width * height * depth; }  
}
```

Arrays

For-each Loop

Command - Line Arguments

Procedural Programming

Object & Class

Class Outline

Methods

Constructor

Garbage Collection

this keyword

Method Overloading

Constructor Overloading

Passing Object to Constructor

Argument Passing

Recursion

Static Members

Array of Objects

Array as Argument

Returning array from Method

Passing Object to Constructor

```
class Box {  
    double width, height, depth;  
    Box(Box ob) {  
        width = ob.width;  
        height = ob.height;  
        depth = ob.depth;  
    }  
    Box(double w, double h, double d){  
        width=w; height=h; depth=d;  
    }  
}
```

- Box mybox1 = new Box(10, 20, 15);
- Box mybox2 = new Box(mybox1);

Arrays

For-each Loop

Command - Line Arguments

Procedural Programming

Object & Class

Class Outline

Methods

Constructor

Garbage Collection

this keyword

Method Overloading

Constructor Overloading

Passing Object to Constructor

Argument Passing

Recursion

Static Members

Array of Objects

Array as Argument

Returning array from Method

Argument Passing

- Two types of variables:
 - simple types
 - class types
- Two corresponding ways of how the arguments are passed to methods:

- **by value**: a method receives a copy of the original value; parameters of simple types

```
void meth(int i, int j){ i*=2; j/=2; }
```

- **by reference**: a method receives the memory address of the original value, not the value itself; parameters of class types

```
void meth(Test o) { o.a *= 2; o.b /= 2; }
```

[Arrays](#)

[For-each Loop](#)

[Command - Line Arguments](#)

[Procedural Programming](#)

[Object & Class](#)

[Class Outline](#)[Methods](#)[Constructor](#)[Garbage Collection](#)[this keyword](#)[Method Overloading](#)[Constructor Overloading](#)[Passing Object to Constructor](#)

[Argument Passing](#)

[Recursion](#)[Static Members](#)

[Array of Objects](#)

[Array as Argument](#)

[Returning array from Method](#)

Recursion

- A recursive method is a method that calls itself
 - all method parameters and local variables are allocated on the stack
 - arguments are prepared in the corresponding parameter positions
 - the method code is executed for the new arguments
 - upon return, all parameters and variables are removed from the stack
 - the execution continues immediately after the invocation point

```
class Factorial {  
    int fact(int n) {  
        if (n == 1)  
            return 1;  
        return fact(n-1) * n;  
    }  
}
```

Arrays

For-each Loop

Command - Line Arguments

Procedural Programming

Object & Class

Class Outline

Methods

Constructor

Garbage Collection

this keyword

Method Overloading

Constructor Overloading

Passing Object to Constructor

Argument Passing

Recursion

Static Members

Array of Objects

Array as Argument

Returning array from Method

Static Members

Static keyword is used for memory management. It belongs to the class than instance of the class

- **static variable:**
 - *static int a=0;*
 - It can be used to refer the common property of all objects
 - It is a global variable shared by all instances of the class
 - It gets memory only once at the time of class loading
 - It cannot be used within a non-static method
- **static method:**
 - *static void meth() { ... }*
 - It belongs to the class rather than object of a class. It can be invoked without the need for creating an instance of a class
 - It can access static data member & can change its value
 - It can only call static methods & access static variables
 - It cannot refer to **this** or **super** in any way
- **static block:**
 - *static { ... }*
 - This is where the **static variables are initialized**
 - It is executed exactly once, when the class is first loaded

[Arrays](#)[For-each Loop](#)[Command - Line Arguments](#)[Procedural Programming](#)[Object & Class](#)[Class Outline](#)[Methods](#)[Constructor](#)[Garbage Collection](#)[this keyword](#)[Method Overloading](#)[Constructor Overloading](#)[Passing Object to Constructor](#)[Argument Passing](#)[Recursion](#)[Static Members](#)[Array of Objects](#)[Array as Argument](#)[Returning array from Method](#)

Array of Objects

- **classname[] arrayname=new classname[no of objects];**

child [] children=new child[5];

- **objectname[objectnumber]=new classname(age);**

children[0]=new child(23);

[Arrays](#)

[For-each Loop](#)

[Command - Line Arguments](#)

[Procedural Programming](#)

[Object & Class](#)

[Class Outline](#)

[Methods](#)

[Constructor](#)

[Garbage Collection](#)

[this keyword](#)

[Method Overloading](#)

[Constructor Overloading](#)

[Passing Object to Constructor](#)

[Argument Passing](#)

[Recursion](#)

[Static Members](#)

[Array of Objects](#)

[Array as Argument](#)

[Returning array from Method](#)

Array of Objects...

Array of Objects...

```
class Child{
    int age;
    Child(int a){age=a;}
    void display(){
        System.out.println("Age"+age);
    }
}

class Test{
    public static void main(String []args){
        Child []ch=new Child[3];
        ch[0]=new Child(20);
        ch[1]=new Child(21);
        ch[2]=new Child(22);
        for(int i=0;i<3; i++){
            ch[i].display();
        }
    }
}
```

Arrays

For-each Loop

Command - Line
Arguments

Procedural
Programming

Object & Class

Class Outline

Methods

Constructor

Garbage Collection

this keyword

Method Overloading

Constructor Overloading

Passing Object to
Constructor

Argument Passing

Recursion

Static Members

Array of Objects

Array as Argument

Returning array from
Method

Array as Argument

- When passing an array to a method, the reference of the array is passed to the method
- Array can be passed to method similar to passing primitive type
- To pass an array as argument, specify the array name without any square bracket within a method call
- As array reference is passed to the method, the corresponding parameter in the called method header must be of array type

[Arrays](#)

[For-each Loop](#)

[Command - Line Arguments](#)

[Procedural Programming](#)

[Object & Class](#)

[Class Outline](#)[Methods](#)[Constructor](#)[Garbage Collection](#)[this keyword](#)[Method Overloading](#)[Constructor Overloading](#)[Passing Object to Constructor](#)[Argument Passing](#)[Recursion](#)[Static Members](#)

[Array of Objects](#)

[Array as Argument](#)

[Returning array from Method](#)

Array as Argument...

Array as Argument...

```
class Test{
    public static void main(String []args){
        int [] data={40,50, 10, 30, 20, 5};
        System.out.println("Unsorted Array:");
        disp(data);
        sort(data);
        System.out.println("Sorted Array:");
        disp(data);
    }
    static void disp(int num[]){
        for(int i=0;i<num.length; i++){
            System.out.print(num[i]+" ");
        }
    }
    static void sort(int num[]){
        int i,j, temp;
        for(i=0; i<num.length-1; i++){
            for(j=0; j<num.length-i-1; j++){
                if(num[j]>num[j+1]){
                    temp=num[j];
                    num[j]=num[j+1];
                    num[j+1]=temp;
                }
            }
        }
    }
}
```

Arrays

For-each Loop

Command - Line Arguments

Procedural Programming

Object & Class

Class Outline

Methods

Constructor

Garbage Collection

this keyword

Method Overloading

Constructor Overloading

Passing Object to Constructor

Argument Passing

Recursion

Static Members

Array of Objects

Array as Argument

Returning array from Method

Returning array from Method

Returning array from Method

When a method returns an array, the reference of the array is returned

```
public class Test{
    public static void main(String []args){
        int [] data={40,50, 10, 30, 20, 5};
        System.out.println("Original Array:");
        disp(data);
        int []rdata=rev(data);
        System.out.println("Reversed Array:");
        disp(rdata);
    }
    static void disp(int num[]){
        for(int i=0;i<num.length; i++)
            System.out.print(num[i]+" ");
    }
    static int [] rev(int num[]){
        int []rdata=new int[num.length];
        for(int i=0, j=rdata.length-1; i<num.length; i++, j--)
            rdata[j]=num[i];
        return rdata;
    }
}
```

Arrays

For-each Loop

Command - Line Arguments

Procedural Programming

Object & Class

Class Outline

Methods

Constructor

Garbage Collection

this keyword

Method Overloading

Constructor Overloading

Passing Object to Constructor

Argument Passing

Recursion

Static Members

Array of Objects

Array as Argument

Returning array from Method