

More sophisticated behaviour

Lecture 09

Waterford Institute of Technology

February 22, 2016

John Fitzgerald

Presentation outline

Estimated duration presentation

Questions at end presentation

Topics discussed:

- The Java Library
- Application Programming Interface (API)
- Generating random numbers
- Collection framework: HashMap, HashSet
- Two-dimensional arrays
- Anonymous objects
- Chaining (fluent programming)
- More control flow techniques: switch, break, continue
- Enum type
- Formatting output

The Java Library

Java library contains thousands of classes

Become familiar with small, frequently used subset

- Classes already encountered
 - String
 - ArrayList
- Classes explored in this session
 - Random
 - HashMap
 - HashSet

The Java Library

Overview Java Platform Standard Edition 7

Java™ Platform
Standard Ed. 7

All Classes

Packages

[java.applet](#)
[java.awt](#)
[java.awt.color](#)
[java.awt.datatransfer](#)
[java.awt.dnd](#)
[java.awt.event](#)
[java.awt.font](#)
[java.awt.geom](#)
[java.awt.im](#)
[java.awt.im.spi](#)
[java.awt.image](#)
[java.awt.image.renderable](#)
[java.awt.print](#)
[java.beans](#)
[java.beans.beancontext](#)

Download

All Classes

[AbstractAction](#)
[AbstractAnnotationValueVisitor6](#)
[AbstractAnnotationValueVisitor7](#)
[AbstractBorder](#)
[AbstractButton](#)
[AbstractCellEditor](#)
[AbstractCollection](#)
[AbstractColorChooserPanel](#)
[AbstractDocument](#)

Overview Package Class Use Tree Deprecated Index Help

Prev Next Frames No Frames

Java™ Platform, Standard Edition 7

API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: [Description](#)

Packages

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.

Importing Java packages

Use `import` qualified—`class`—name

Example

- `import java.util.ArrayList;`
- `import java.util.Random;`

Also could use package name but disadvantage possibly thousands classes imported

- `import package—name*;`
- `import java.util*;`

Best be specific

- `import java.util.Date;`
- `import java.util.Random;`

Generics documentation

Parameterized or Generic classes

- Class `ArrayList<E>`
 - Array containing objects class type `E`
 - `E` specified when `ArrayList` variable declared
- Class `HashMap<K, V>`
 - `K` key-type & `V` mapped value type specified when `HashMap` variable declared

```
public class GenericsDemo
{
    private ArrayList<String> notes = new ArrayList<String>();
    private HashMap<String,String> contacts = new HashMap<String,String>()
    ;
    public void generics(){
        notes.add("Mustn't forget to call supervisor");
        contacts.put("Abamo Patrick", "(412) 9888 5467");
    }
}
```

Class components

Class may be described as comprising

- Interface
 - Facilitates third party usage
- Implementation
 - Hidden from user
 - Generally user has no interest

```
public class Square
{
    int size;
    public Square(int size) { //interface
        //implementation
        this.size = size;
    }
    public int getArea() { //interface
        //implementation
        return size*size;
    }
}
```

Class API

Application Programming Interface

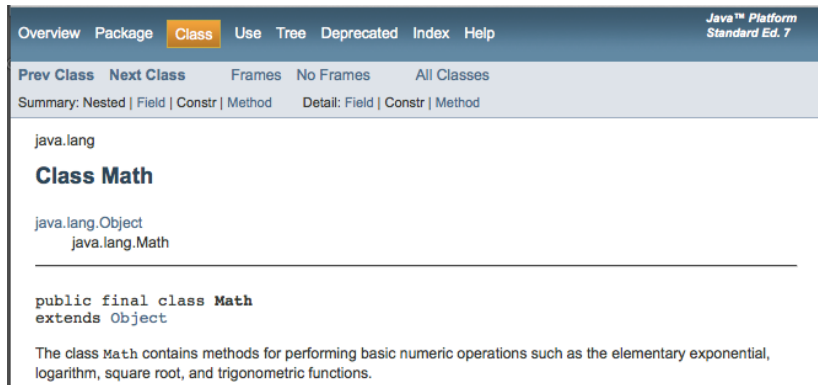
Comprises following class information

- Name
- General description of purpose
- List constructors
- List methods
- Parameters of constructors and methods
- Return types methods
- Description purpose each constructor and method

Class API

Application Programming Interface

Class name and description of purpose



The screenshot shows the Java Platform Standard Ed. 7 Class API page for the `java.lang.Math` class. The page has a dark blue header with navigation links: Overview, Package, Class (highlighted), Use, Tree, Deprecated, Index, and Help. Below the header, there are links for Prev Class, Next Class, Frames, No Frames, and All Classes. The main content area displays the package `java.lang` and the class `Math`. It also shows the inheritance hierarchy: `java.lang.Object` and `java.lang.Math`. The class declaration is shown as `public final class Math extends Object`. A description follows: "The class `Math` contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions."

Overview Package **Class** Use Tree Deprecated Index Help

Java™ Platform
Standard Ed. 7

Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

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.

Class API

Application Programming Interface

List constructors and their parameters

Constructor Summary

Constructors

Constructor and Description
String() Initializes a newly created <code>String</code> object so that it represents an empty character sequence.
String(byte[] bytes) Constructs a new <code>String</code> by decoding the specified array of bytes using the platform's default charset.
String(byte[] bytes, Charset charset) Constructs a new <code>String</code> by decoding the specified array of bytes using the specified <code>charset</code> .
String(byte[] ascii, int hibyte) Deprecated. <i>This method does not properly convert bytes into characters. As of JDK 1.1, the preferred way to do this is via the <code>String</code> constructors that take a <code>Charset</code> charset name, or that use the platform's default charset.</i>

Class API

Application Programming Interface

List of fields

Field Summary

Fields

Modifier and Type	Field and Description
static double	E The double value that is closer than any other to e, the base of the natural logarithms.
static double	PI

Class API

Application Programming Interface

The methods

Method Summary

Methods

Modifier and Type	Method and Description
static double	<code>abs(double a)</code> Returns the absolute value of a double value.
static float	<code>abs(float a)</code> Returns the absolute value of a float value.
static int	<code>abs(int a)</code> Returns the absolute value of an int value.

Class API

Application Programming Interface

Detailed information about the fields

Field Detail

E

```
public static final double E
```

The double value that is closer than any other to e , the base of the natural logarithms.

See Also:

[Constant Field Values](#)

Class API

Application Programming Interface

Detailed information about the methods

Method Detail

sin

```
public static double sin(double a)
```

Returns the trigonometric sine of an angle. Special cases:

- If the argument is NaN or an infinity, then the result is NaN.
- If the argument is zero, then the result is a zero with the same sign as the argument.

The computed result must be within 1 ulp of the exact result. Results must be semi-monotonic.

Parameters:

Interface of Method

Interface terminology applicable to methods

Comprises

- Signature (name & parameters)
- Return type
- Descriptive comments (example Since 1.6)

isEmpty

```
public boolean isEmpty()
```

Returns true if, and only if, [length\(\)](#) is 0.

Returns:

true if [length\(\)](#) is 0, otherwise false

Since:

1.6

Randomness

Generate pseudo random array numbers

Two commonly used approaches:

- `Math.random()` returns *double* in range $[0, 1)$
- `Random` class simpler to use

```
//Generate random int in range 2 to 8 inclusive  
int rndNmr = (int)(Math.random()*7 + 2)
```

```
// Generate random int in range 0 to 6 inclusive  
Random rnd = new Random();  
int rndNmr = rnd.nextInt(7); //in range 0 to 6 inclusive  
rndNmr += 2; //now in range 2 to 8 inclusive  
// Less verbose  
int rndNmr = new Random().nextInt(7) + 2;
```


Java Random class

Objects of Random class can

- Generate pseudorandom number stream
- In range
 - Integer.MIN_VALUE to Integer.MAX_VALUE
 - -2147483648 to 2147483647

```
import java.util.Random;  
Random randomGenerator = new Random();  
//Generated randNmr is in range -2147483648 to 2147483647  
int randNmr = randomGenerator.nextInt();  
System.out.println(randNmr);  
//Generated randNmr2 is in range 0 to n-1 inclusive  
int randNmr2 = randomGenerator.nextInt(n);
```

Java Random class

Generate random integer range

```
//Randomly generate integers range [0, 10)
Random rnd = new Random(10);
for (int i = 0; i < 10; i += 1) {
    int randomNumber = rnd.nextInt(10);
    System.out.println("nextInt [0,10) " +
        randomNumber);
}
```

```
nextInt [0,10) 5
nextInt [0,10) 0
nextInt [0,10) 6
nextInt [0,10) 8
nextInt [0,10) 8
nextInt [0,10) 8
nextInt [0,10) 8
nextInt [0,10) 5
nextInt [0,10) 6
nextInt [0,10) 8
```

Java Random class

Generate random integer range

```
//Randomly generate integers range [20, 30)
Random rnd = new Random(10);
for (int i = 0; i < 10; i += 1) {
    int randomNumber = rnd.nextInt(10) + 20;
    System.out.println("nextInt [20, 30) " +
        randomNumber);
}
```

```
nextInt [20,30) 23
nextInt [20,30) 22
nextInt [20,30) 29
nextInt [20,30) 28
nextInt [20,30) 22
nextInt [20,30) 20
nextInt [20,30) 20
nextInt [20,30) 23
nextInt [20,30) 21
nextInt [20,30) 26
```

HashMap

HashMap object that maps keys to values

- Iteration ordering not guaranteed
- Cannot contain duplicate keys
- Each key maps to at most one value
- Has methods such as
 - `put(K key, V value)`
 - `get(Object key)`
 - `containsKey(Object value)`
 - `remove(Object key)`

```
import java.util.HashMap;  
  
contacts.put("DCU", "(353) 1 8658934");  
String phoneNumber = contacts.get("DCU");  
boolean hasKey = contacts.containsKey("DCU");  
contacts.remove("DCU");
```

HashMap

Example 1

```
import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;
//prints values
HashMap<String, String> contacts = new HashMap<String, String>();
contacts.put("George", "0231 8542983");
contacts.put("Michael", "0595 848290");

Collection<String> c = contacts.values();
Iterator<String> it = c.iterator();
while (it.hasNext())
    System.out.println(it.next());
```

HashMap

Example 2

```
//prints key-value pairs
HashMap<String, String> contacts = new HashMap<String, String>();
contacts.put("George", "0231 8542983");
contacts.put("Michael", "0595 848290");

Collection<String> k = contacts.keySet();
Iterator<String> it2 = k.iterator();
while (it2.hasNext())
{
    String key = (String)it2.next();
    String val = contacts.get(key);
    System.out.println("key " + key + " value " + val);
}
```

HashSet

HashSet object has collection distinct elements

- Iteration ordering not guaranteed
- Cannot contain duplicate elements
- Has methods such as
 - `add(E e)`
 - `contains(Object o)`
 - `remove(Object o)`

```
import java.util.HashSet;  
  
names.add("DCU");  
names.add("DCU");//ignored  
names.contains("DCU");  
names.remove("DCU");
```

Arrays

Two dimensional

As with one-dimensional arrays:

- Stores fixed number of elements
- All values same type
- Size fixed at creation

Example creation and initialization 2-d array:

```
int nmrRows = 3;
int nmrCols = 4;
int[][] ar2d = new int[nmrRows][nmrCols];
for(int row = 0; row < nmrRows; row += 1)
{
    for(int col = 0; col < nmrCols; col += 1)
    {
        ar2d[row][col] = row + col;
    }
}
```

0	1	2	3
1	2	3	4
2	3	4	5

Arrays

Two dimensional

Rows may be different lengths

- Each row a one-dimensional array

Example 2-d array variable row lengths:

```
int nmrRows = 3;
int nmrCols = 4;
int[][] ar2d = new int[nmrRows][];
for(int row = 0; row < nmrRows; row += 1)
{
    ar2d[row] = new int[nmrCols + row];
    for(int col = 0; col < ar2d[row].length; col += 1)
    {
        ar2d[row][col] = row + col;
    }
}
```

0	1	2	3			
1	2	3	4	5		
2	3	4	5	6	7	

Anonymous objects

Use of anonymous objects common idiom

```
//Verbose
public class College
{
    private String student;

    public College()
    {
        Student student = new Student();
        setState(student);
    }

    public void setState(Student student)
    {
        this.student = student;
    }
}
```

```
//Use anonymous object
public class College
{
    private String student;

    public College()
    {
        setState(new Student());
    }

    public void setState(Student
        student)
    {
        this.student = student;
    }
}
```

Chaining

Fluent programming

Consider this verbose style

```
Student student = new Student();  
student.setName("Jane Doe");  
student.setAge(21);  
student.setCourse("Mathematics");
```

```
public class Student {  
    private String name;  
    private int age;  
    private String course;  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public void setAge(int age) {  
        this.age = age;  
    }  
  
    public void setCourse(String course) {  
        this.course = course;  
    }  
}
```

Chaining

Fluent programming

Using chaining:

```
Student student = new Student();  
student.setName("Jane Doe")  
    .setAge(21)  
    .setCourse("Mathematics");
```

```
public class Student {  
    private String name;  
    private int age;  
    private String course;  
  
    public Student setName(String name) {  
        this.name = name;  
        return this;  
    }  
  
    public Student setAge(int age) {  
        this.age = age;  
        return this;  
    }  
  
    public Student setCourse(String course  
        ) {  
        this.course = course;  
        return this;  
    }  
}
```

Control flow

The switch statement

switch statement

- **expression** must resolve to one of char, int, byte, short (or corresponding wrappers), String, Enum

```
switch(expression)
{
    case value 1: statement(s) 1
                  break;
    case value 2: statement(s) 2
                  break;
    ...
    case value n: statement(s) n
                  break;
    default: default statement(s)
}
```

Control flow

The switch statement

switch statement

- Can have number execution paths
- Execution route depends on value of variable or expression

```
int day = 6;
String sDay;
switch(day)
{
    case 6:
    case 7: sDay = "a weekend day";
           break;
    default: sDay = "a work day";
           break;
}
//Outputs: Today is a weekend day
System.out.println("Today is " + sDay);
```

Control flow

The *break* statement

- Exits *switch* statement
- Terminates *for*, *while*, *do-while* loop
- May be labelled or unlabelled

//Example unlabelled *break*

```
int[] arInt = {10, 20, 30, 40, 50, 60};
int searchNmr = 30;
for(int i = 0; i < arInt.length; i += 1) {
    if(arInt[i] == searchNmr) {
        System.out.println("Found it");
        break;
    }
}
```

Control flow

Labelled *break* statement

- Unlabelled: exits inner block *for* loop
- Labelled: exits outer block *for* loop

```
int searchfor = 12;
search :
for (int i = 0; i < arrayOfInts.length; i++) {
    for (int j = 0; j < arrayOfInts[i].length; j++) {
        if (arrayOfInts[i][j] == searchfor) {
            foundIt = true;
            break search;
        }
    }
}
```


Control flow

Refactor to avoid **break**

- Return *true* immediately if *searchFor* found
- Else if not found, return *false*

```
boolean isFound(int[][] arrayOfInts, int searchFor)
{
    for (int i = 0; i < arrayOfInts.length; i++) {
        for (int j = 0; j < arrayOfInts[i].length; j++) {
            if (arrayOfInts[i][j] == searchFor)
                return true;
        }
    }
    return false;
}
```

Control flow

The *continue* statement a very contrived example

continue statement

- Skips current iteration *for*, *while*, *do-while* loop
- Unlabelled form
 - skips to end innermost loop's body
 - evaluates boolean expression controlling loop

//Outputs 6. Comment out *continue*: outputs 27

```
String searchMe = "picked peck pickled peppers";
int max = searchMe.length();
int numberPs = 0;

for (int i = 0; i < max; i++) {
    // only count when p found
    if (searchMe.charAt(i) != 'p') {
        continue;
    }
    numberPs++;
}

System.out.println("Found " + numberPs + " p's in the string.");
```

Enum

Special data type

Enumerated type comprises

- outer wrapper similar to class
- but uses *enum* keyword
- and body is list variable names
- that denote values relating to type

```
public enum Group
{
    // Categories social networks
    FRIENDS, ENEMIES, FAMILY, WORK;
}
```

```
// Example using an enum
Group group = getGroup(){...}
if (group == Group.FRIENDS) {...}
```

Enum

Special data type

```
enum Day {WEEKDAY, WEEKEND}  
public static void makePlans(Day day) {  
    switch (day) {  
        case WEEKDAY:  
            System.out.println("Working like a dog;");  
            break;  
        case WEEKEND:  
            System.out.println("Sleeping like a log");  
            break;  
        default:  
    }  
}
```

```
makePlans(Day.WEEKDAY); // Output: Working like a dog  
makePlans(Day.WEEKEND); // Output: Sleeping like a log
```

Output

Formatter

```
public class FormatterExample
{
    public static void main(String[] args)
    {
        // sample data
        double[] dcontent = {.0456, 4.3225555, 5.0, -5};
        String[] slabel = {"Shopping", "Sport", "Entertainment", "Savings"};

        // example formatted output
        System.out.println(String.format("%-20s %s", "Label", "Content"));
        System.out.println("-----");
        for (int i = 0; i < dcontent.length; i += 1)
        {
            System.out.printf(String.format("%-20s %5.2f %n", slabel[i], dcontent[i]));
        }
    }
}
```

Label	Content

Shopping	0.05
Sport	4.32
Entertainment	5.00
Savings	-5.00

create new line

placeholder decimal number

placeholder 20 char string

- denotes left justified

Summary

- Java library and importing packages
- Class application programming interface (API)
- Generating (pseudo)random data
- HashMap
- HashSet
- Two dimensional arrays
- Anonymous objects
- Chaining
- Control flow statements: *switch*, *break* and *continue*
- Enum type

Referenced Material

1. Overview Java Platform Standard Edition 7

<http://docs.oracle.com/javase/7/docs/api/>

[Accessed 2014-02-18]

2. String(Java Platform SE7) <http://docs.oracle.com/javase/7/docs/api/java/lang/String.html>

[Accessed 2014-02-19]

3. ArrayList(Java Platform SE 7) <http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html>

[Accessed 2014-02-19]

Referenced Material

4. Random(Java Platform SE 7) <http://docs.oracle.com/javase/7/docs/api/java/util/Random.html>

[Accessed 2014-02-19]

5. HashMap(Java Platform SE 7) <http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html>

[Accessed 2014-02-20]

6. HashSet(Java Platform SE 7) <http://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html>

[Accessed 2014-02-20]

Referenced Material

7. AutoBoxing(Java Platform SE 7)

<http://docs.oracle.com/javase/tutorial/java/data/autoboxing.html> [Accessed 2014-02-24]

8. Enum Types (The Java Tutorials)

<http://docs.oracle.com/javase/tutorial/java/java00/enum.html> [Accessed 2014-02-16]

9. Class Formatter

<http://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html>

[Accessed 2015-03-21]

Referenced Material

10. Labelled and unlabelled **break** statements

`https://docs.oracle.com/javase/tutorial/java/nutsandbolts/branch.html`

[Accessed 2016-02-20]