More sophisticated behaviour
Lecture 10

Waterford Institute of Technology

February 19, 2016

John Fitzgerald

The Java Library

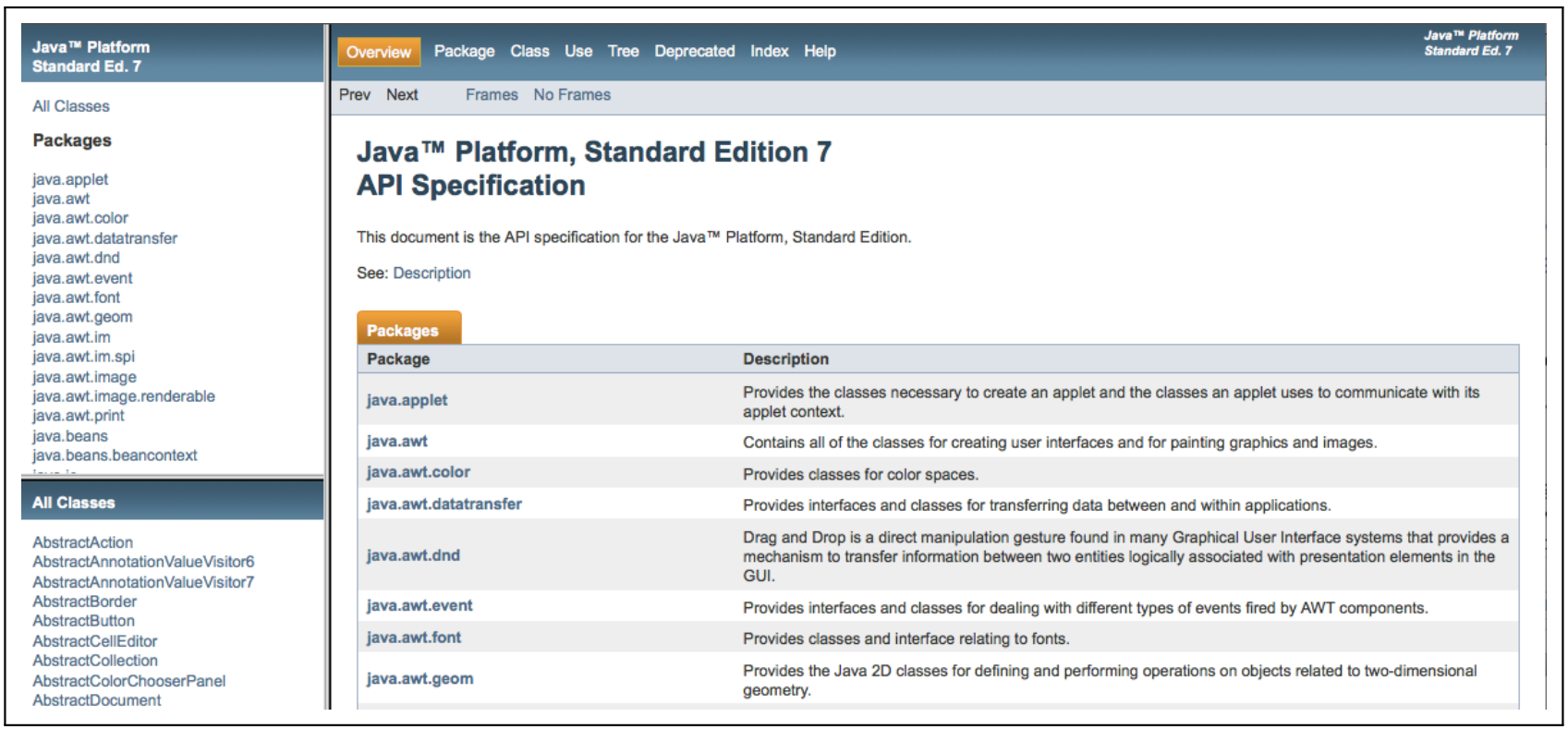
Java library contains thousands of classes

Become familiar with small, frequently used subset

- Classes already encountered
  - String
  - ArrayList
- Classes explored in this session
  - Random
  - HashMap
  - HashSet
  - Arrays (Two dimensional)

The Java Library

Overview Java Platform Standard Edition 7



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

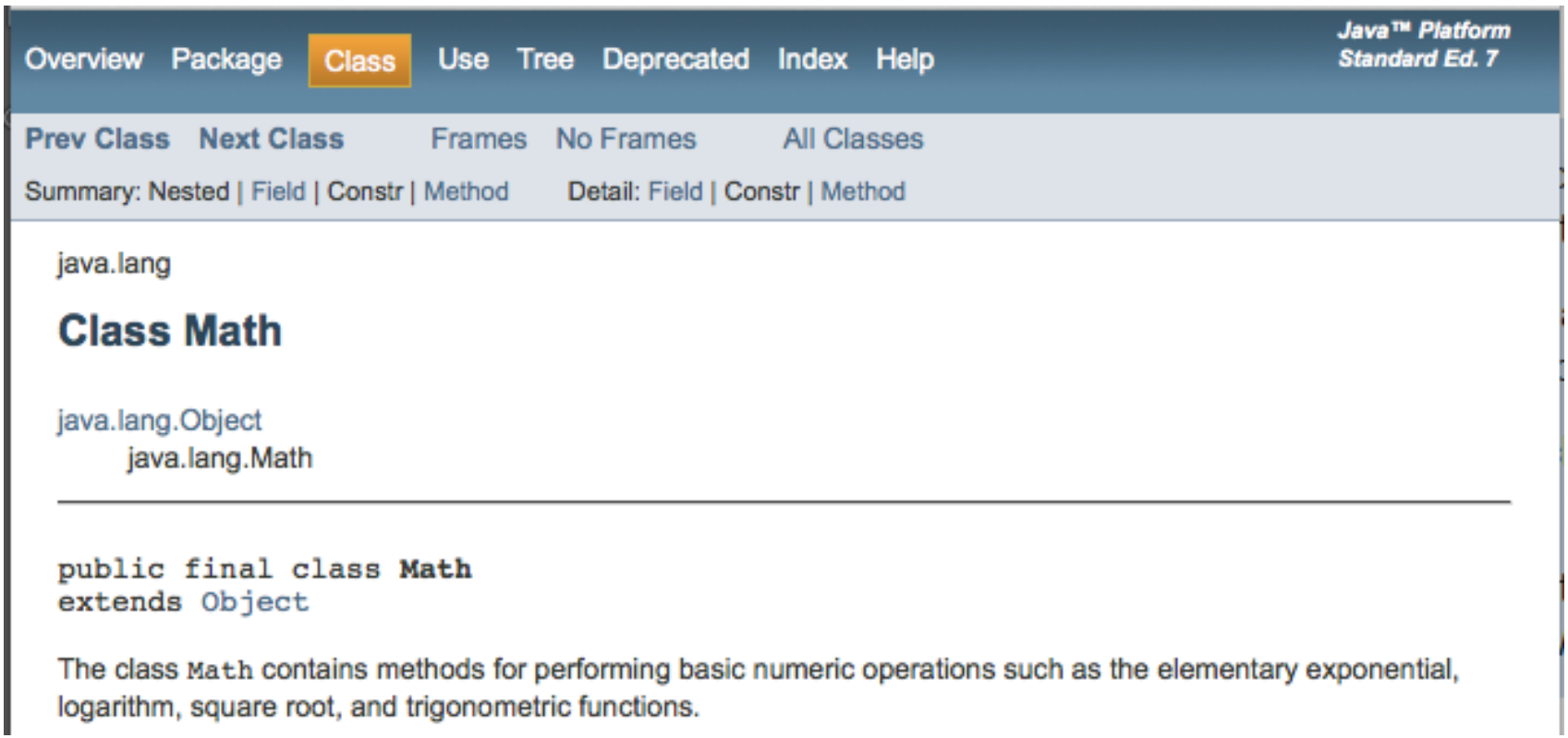
Comprises following class information

- 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

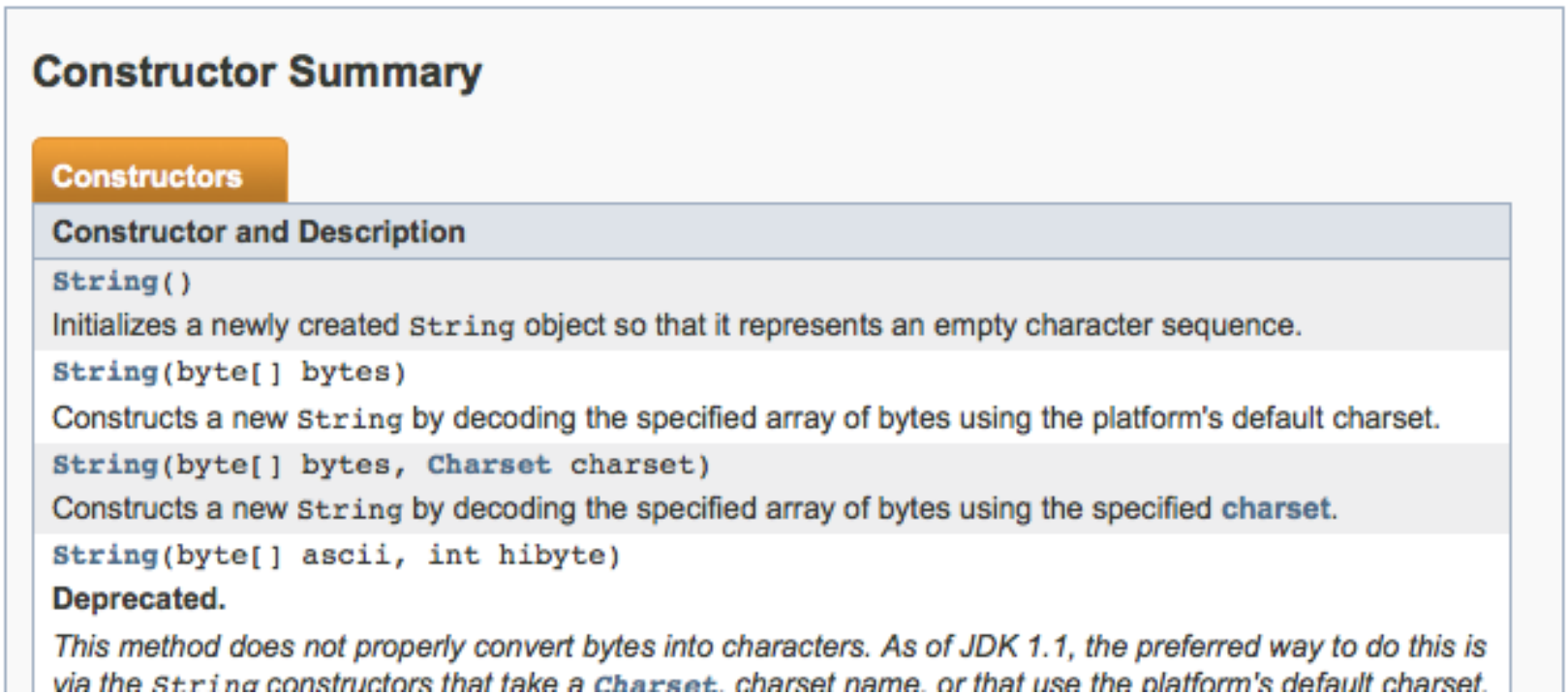
Class name and description of purpose



Class API

Application Programming Interface

List constructors and their parameters



Constructor Summary

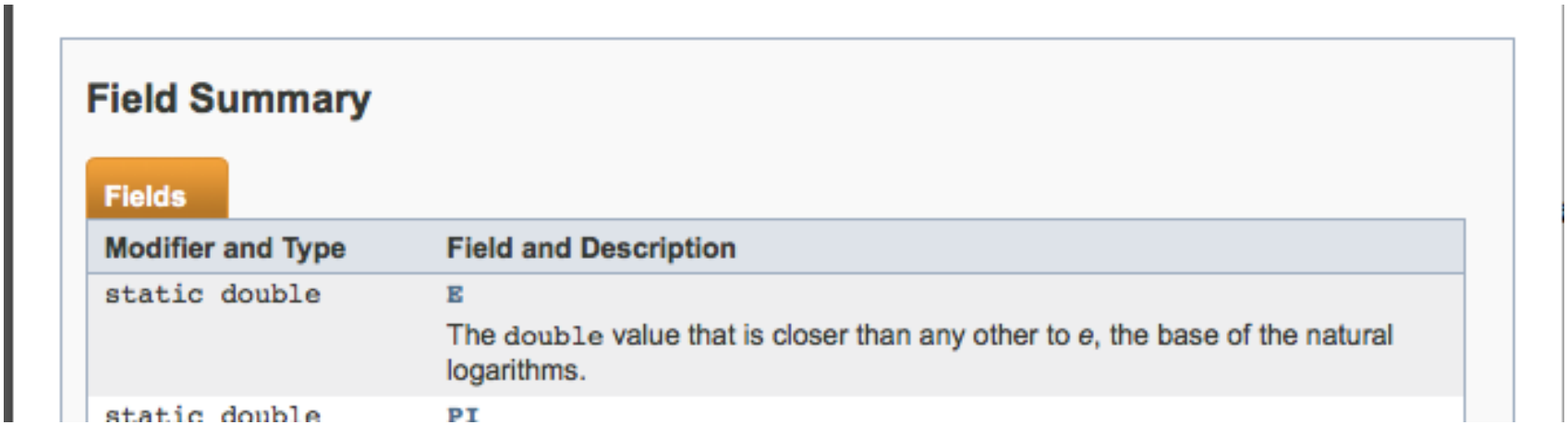
Constructors

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

Class API

Application Programming Interface

List of fields



Field Summary

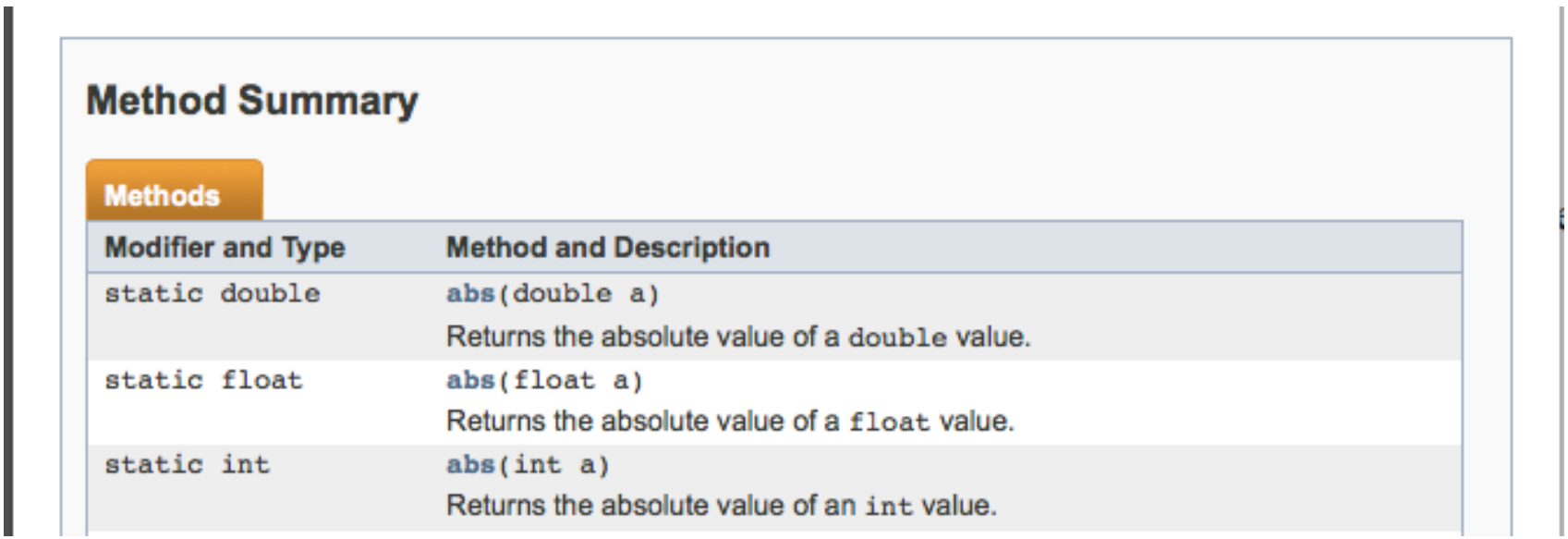
Fields

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

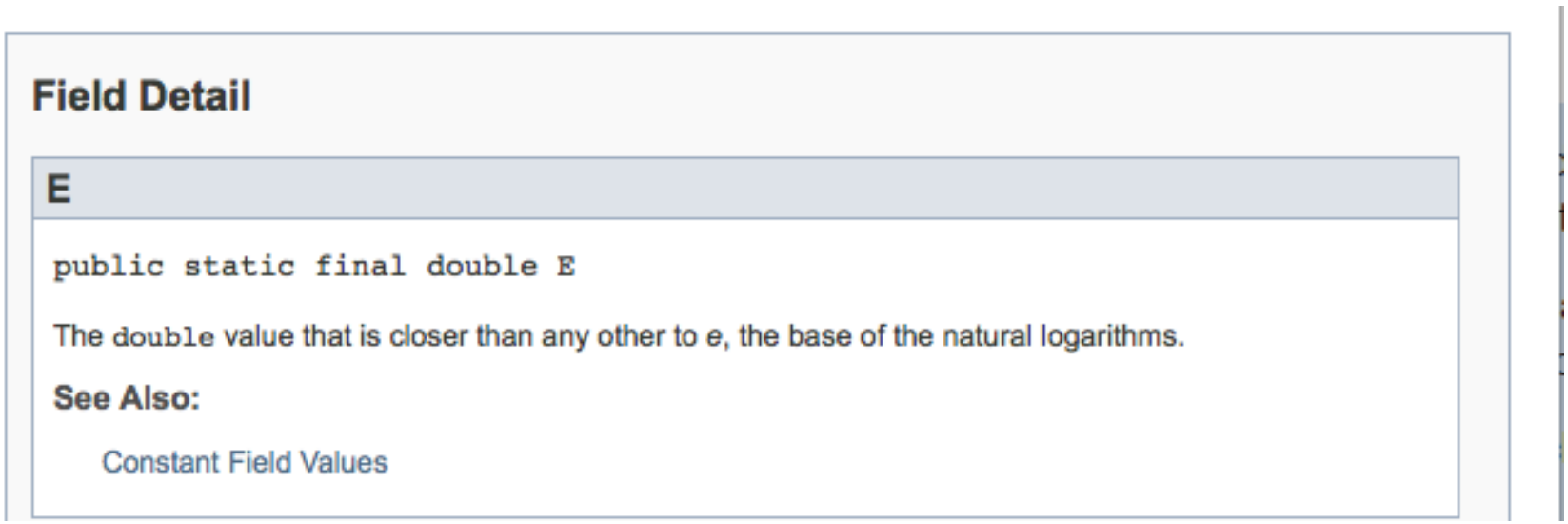
Methods

Modifier and Type	Method and Description
`static double`	`abs(double a)` Returns the absolute value of a `double` value.
`static float`	`abs(float a)` Returns the absolute value of a `float` value.
`static int`	`abs(int a)` 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.

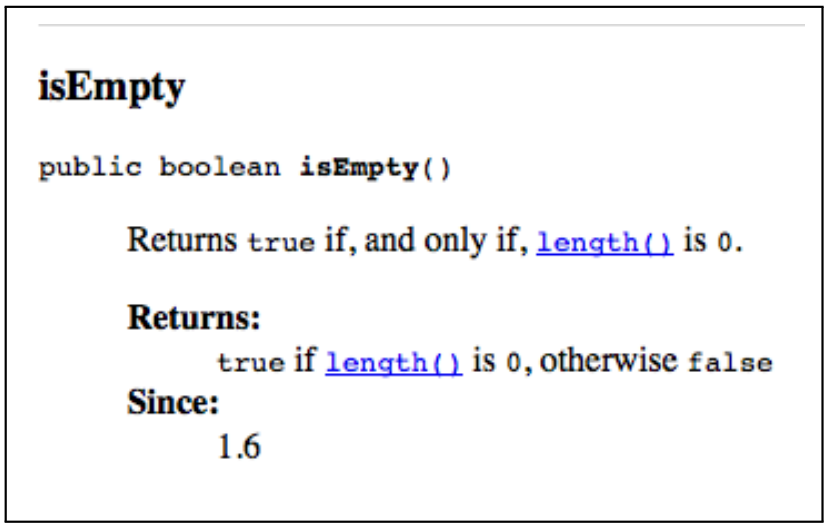
Parameters:

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 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
```

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)
for(int i = 0; i < 10; i += 1)
  System.out.println("nextInt [0,10) " + rnd
      .nextInt(10));
```

```
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)
for(int i = 0; i < 10; i += 1)
  System.out.println("nextInt [0,10) " + (
      rnd.nextInt(10) + 20));
```

```
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 void setName(String name) {
        this.name = name;
        return this;
    }

    public void setAge(int age) {
        this.age = age;
        return this;
    }

    public void setCourse(String course) {
     this.course = course;
     return this;
}
```

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

break statement

- Terminates *for, while, do-while* loop
- Can 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

The *continue* statement

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

Variable selectable from set predefined constants

- `enum Day {WEEKDAY, WEEKEND}`

```
enum Day {WEEKDAY, WEEKEND}
public class EnumTest {
    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:
        }
    }
    public static void main(String[] args) {
        makePlans(Day.WEEKDAY);
    }
}
```

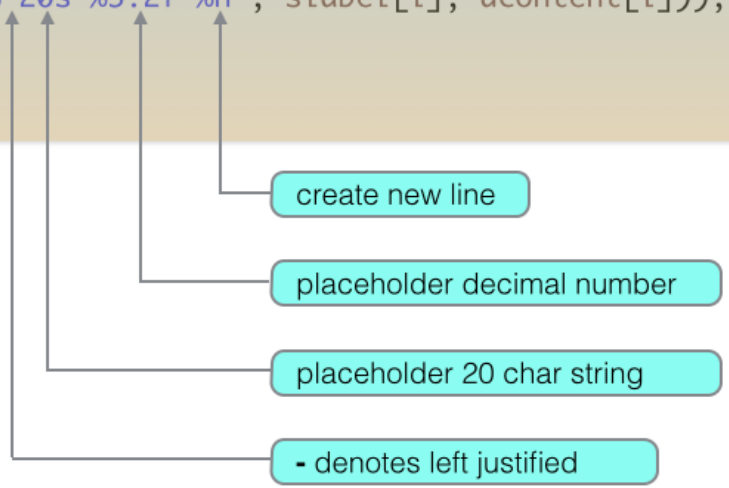
Output

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
```



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/javaOO/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]