COM6516
Object Oriented Programming and Software Design

The contents of this module has been developed by Adam Funk, Kirill Bogdanov, Mark Stevenson, Richard Clayton and Heidi Christensen

Practical 3

Inheritance etc

- Recursion
- `static`: keyword for class fields and class methods
- Inheritance from the `Object` superclass
- Random number generation

Fibonacci series

Fibonacci series: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

- the first two numbers are 1 and 1
- the rest are the sum of the preceding two numbers

This can be implemented using **iteration** or **recursion**

Recursion

```
public class Fib {
    public static int fibr(int n) {
        if ((n == 1) || (n == 2)) {
            return 1;
        }
        else {
            return fibr(n-1) + fibr(n-2);
        }
    }

    public static void main(String[] args) {
        for (int i = 1; i < 40; i++) {
            System.out.println(i + ": " + fibr(i));
        }
    }
}
```

Why recursion is slower than iteration?

The recursive solution is compact, but it does not store previously calculated numbers, hence it needs lots of recursion calculation

(e.g.)

- to calculate fibr(40), it requires to calculate fibr(39) and fibr(38), and
- to calculate fibr(39), it requires to calculate fibr(38) and fibr(37), and
- to calculate fibr(38), it requires to calculate fibr(37) and fibr(36), and
- …

Example: `Math` class

https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/lang/Math.html

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

Field summary

`static double E`

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

Method summary

`static double exp(double a)`

returns Euler's number *e* raised to the power of a double value

Class methods

Example: class vs instance methods …

- Because `exp` is a class method in the `Math` class:

```
float eSquared = Math.exp(2);
```

- Suppose `exp` was not a class method:

```
Math mathObject = new Math();
float eSquared = mathObject.exp(2);
```

Consider class methods when …

- you are writing utility classes
- method does not any instance field
- operation does not rely on instance creation
- some code can be shared by many instance methods
- definition of the method will never be changed or overridden

Inheritance from the `Object` superclass

`toString` method of the `Circle` class:

```
public class Circle {
    …
    public String toString() {
        return "circle with radius " + this.radius;
    }
    …
}
```

Using the default `toString` method from the `Object` superclass:

```
Circle@6ff3c5b5
```

Using the `toString` method from the `Circle` class above, that overrides the default method from the superclass:

```
circle with radius 3.0
```

Inheritance from the `Object` superclass

`equals` method of the `Circle` class:

```
public boolean equals(Object other) {
    if (this == other) {
        return true;
    }
    if (other == null) {
        return false;
    }
    if (this.getClass() != other.getClass()) {
        return false;
    }
    return (this.radius == ((Circle) other).radius);
}
```

Using the default `equals` method from the `Object` superclass:

```
myCircle2 equals myCircle false
```

Using the `equals` method from the `Circle` class above, that overrides the default method from the superclass:

```
myCircle2 equals myCircle true
```

Inheritance from the `Object` superclass

`equals` method of the `Circle` class:

```
public boolean equals(Object other) {
    if (this == other) {
        return true;
    }
    if (other == null) {
        return false;
    }
    if (this.getClass() != other.getClass()) {
        return false;
    }
    return (this.radius == ((Circle) other).radius);
}
```

To test the **identity** (do the object references point to the same memory location?):

```
System.out.println("myCircle2 == myCircle " + (myCircle2 == myCircle));
```

To test the **equality** (are the object states the same?):

```
System.out.println("myCircle2 equals myCircle "
                   + myCircle2.equals(myCircle));
```

Inheritance from the `Publication` class

`Book` class constructor:

```
public Book(String t, String a, int i, int n, int c) {
    super(t, a, i, n);  // parameterised superclass constructor
    numChapters = c;
}
…
public static void main(String[] args) {
    Book myBook = new Book("Core Java", "Horstmann", 968918, 985, 12);
    System.out.println(myBook);
}
```

Parameters are set by using the superclass constructor:

```
Book[title="Core Java",author="Horstmann",isbn=968918,numPages=985]
[numChapters=12]
```

By removing `super(t, a, i, n)` from the above, parameters are not set:

```
Book[title="default",author="default",isbn=0,numPages=0]
[numChapters=12]
```