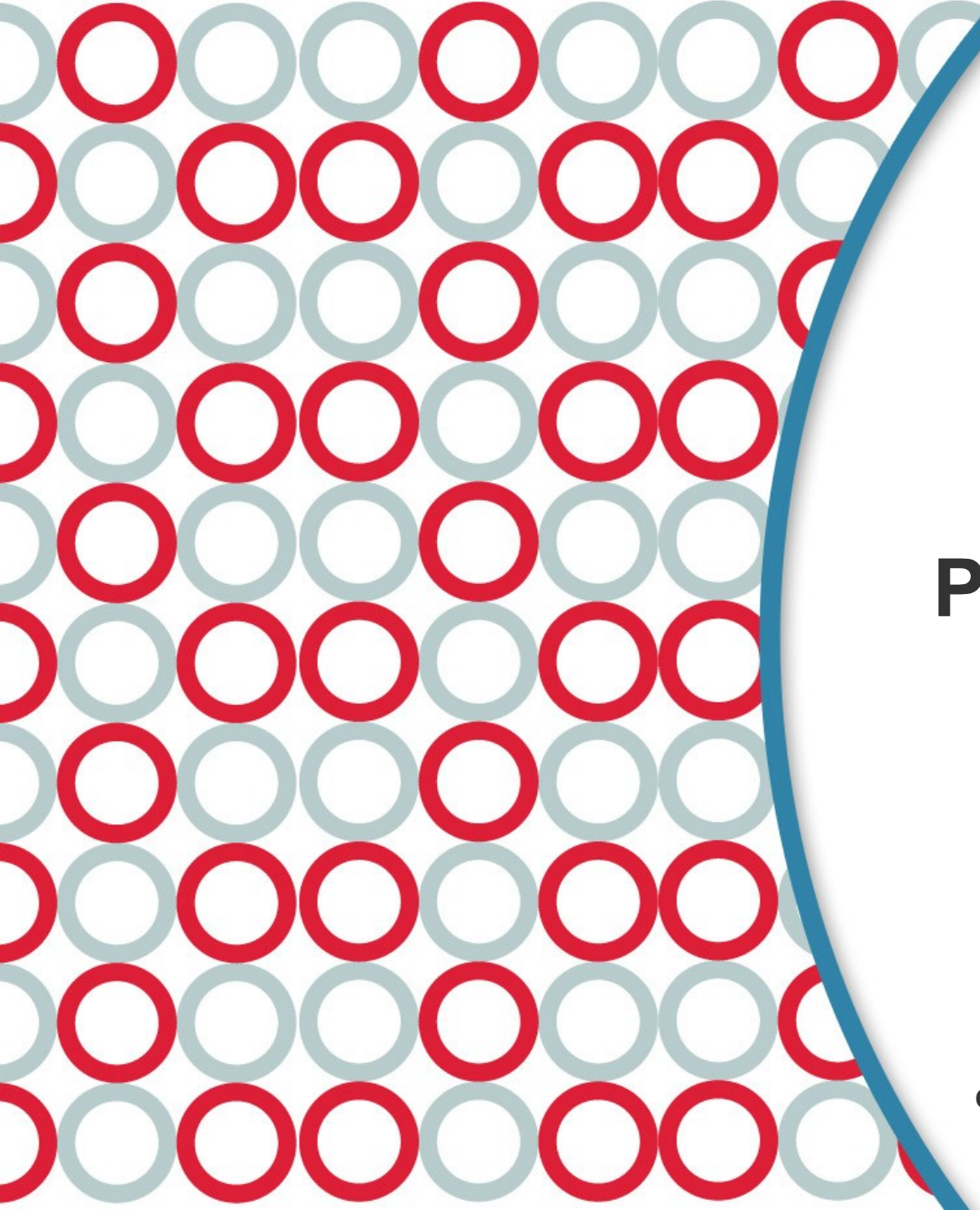


February 2022,
Programming in Java

Programming in Java

Trayan Iliev
tiliev@iproduct.org
http://iproduct.org

About me

Trayan Iliev

– CEO of IPT – Intellectual Products & Technologies

– Oracle® certified programmer 15+ Y

– end-to-end reactive fullstack apps with Java, ES6/7, TypeScript, Angular, React and Vue.js

– 12+ years IT trainer

– Voxxed Days, jPrime, jProfessionals, BGOUG, BGJUG, DEV.BG speaker

– Organizer RoboLearn hackathons and IoT enthusiast

Where to Find the Code?

Java Web Development projects and examples are available @ GitHub:

https://github.com/iproduct/java-fundamentals-2022

Agenda for This Session

- Java Class structure – package, imports, fields, methods, access modifiers;
- Creating objects – constructors, order of initialization, static members, keyword this, constructors overloading;
- Working with methods – designing methods, arguments and return values, overloading, static methods, access modifiers;
- Define the scope of variables – class(static), local, instance variables;
- Apply encapsulation principles to a class;
- Understand objects equality – the difference between “==” and equals();
- Wrapper Classes;
- Distinguish between Object reference and primitive variables, type casting; Methods reference and primitive arguments;
- Enumerations;
- Object lifecycle – destroying objects, garbage collection – finalize();

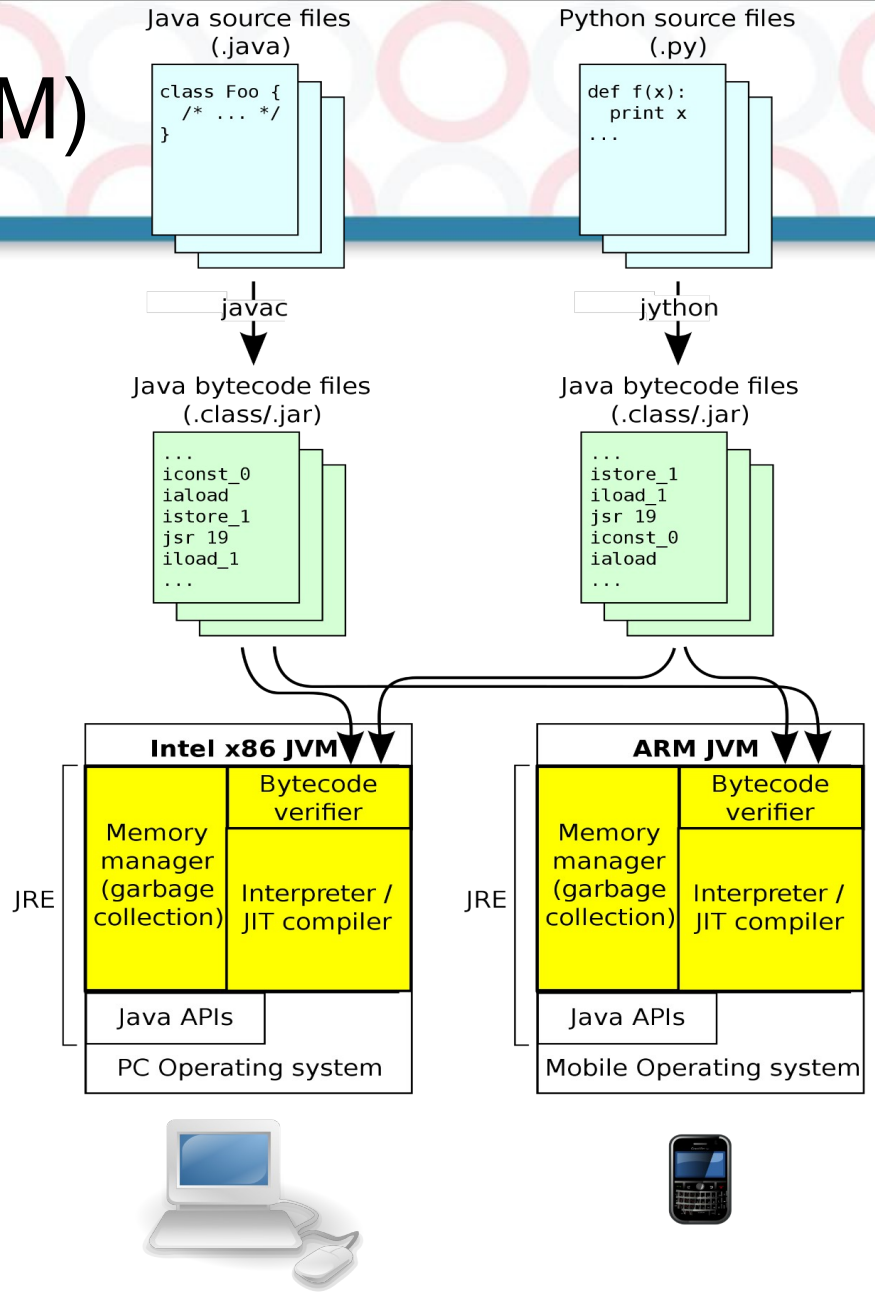
Key Features of Java Language

- Single base hierarchy - inheritance from only one parent class, with the possibility of implementation of multiple interfaces
- Garbage Collector – portability and platform independence, fewer errors
- Secure Code – separation of business logic from the error handling and exceptions
- Multithreading - easy realization of parallel processing
- Persistence – Java Database Connectivity (JDBC) and Java Persistence API (JPA)

Integrated Development Environments for Java Applications

- Java™ development environment types:
- JavaSE, JavaEE, JavaME, JavaFX
- JavaSE: Java Development Kit (JDK) and Java Runtime Environment (JRE)
- Java™ compiler - javac
- Java Virtual Machine (JVM) - java
- Sourse code → Byte code
- Installing JDK 8+
- Compile and run programs from the command line
- IDEs: IntelliJ IDEA, Eclipse

Java Virtual Machine(JVM)



Java Application Stack

Java™ Custom Application – Level & patterns of garbage production, Concurrency, IO/Net, Algorithms & Data structures, API & Frameworks

Application Server – Web Container, EJB Container, Distributed Transactions Dependency Injection, Persistence - Connection Poolling, Non-blocking IO

Java™ Virtual Machine (JVM) – Gartbage Collection, Threads & Concurrency, NIO

Operating System – Virtual Memory, Paging, OS Processes and IO/Net libraries

Hardware Platform – CPU, Memory, IO, Network

Processing Node 1 | Processing Node2 | ... | Processing Node N

Level of Optimization

Classes, Objects and References

- Class - set of objects that share a common structure, behaviour and possible links to objects of other classes = objects type
  - ✓ structure = attributes, properties, member variables
  - ✓ behaviour = methods, operations, member functions, messages
  - ✓ relations between classes: association, inheritance, aggregation, composition – modeled as attributes (references to objects from the connected class)
- Objects are instances of the class, which is their addition:
  - ✓ own state
  - ✓ unique identifier = reference pointing towards object
-

Object (Reference) Data Types

- Creating a class (a new data type)

```
class MyClass { /* attributes and methods of the class */ }
```

- Create an object (instance) from the class MyClass :

```
MyClass myObject = new MyClass();
```

- Declaration and initialization of attributes:

```
class Person {
        String name = “Anonimous”;
        int age;
}
```

- Access to attribute: `Person p1 = new Person();`

```
p1.name = “Ivan Petrov”;        p1.age = 28;
```

Creating Objects

- Class String – modeling string of characters:

– declaration:

```
String s;
```

– initialization (on separate line):

```
s = new String("Hello Java World");
```

– declaration + initialization:

```
String s = new String("Hello Java World");
```

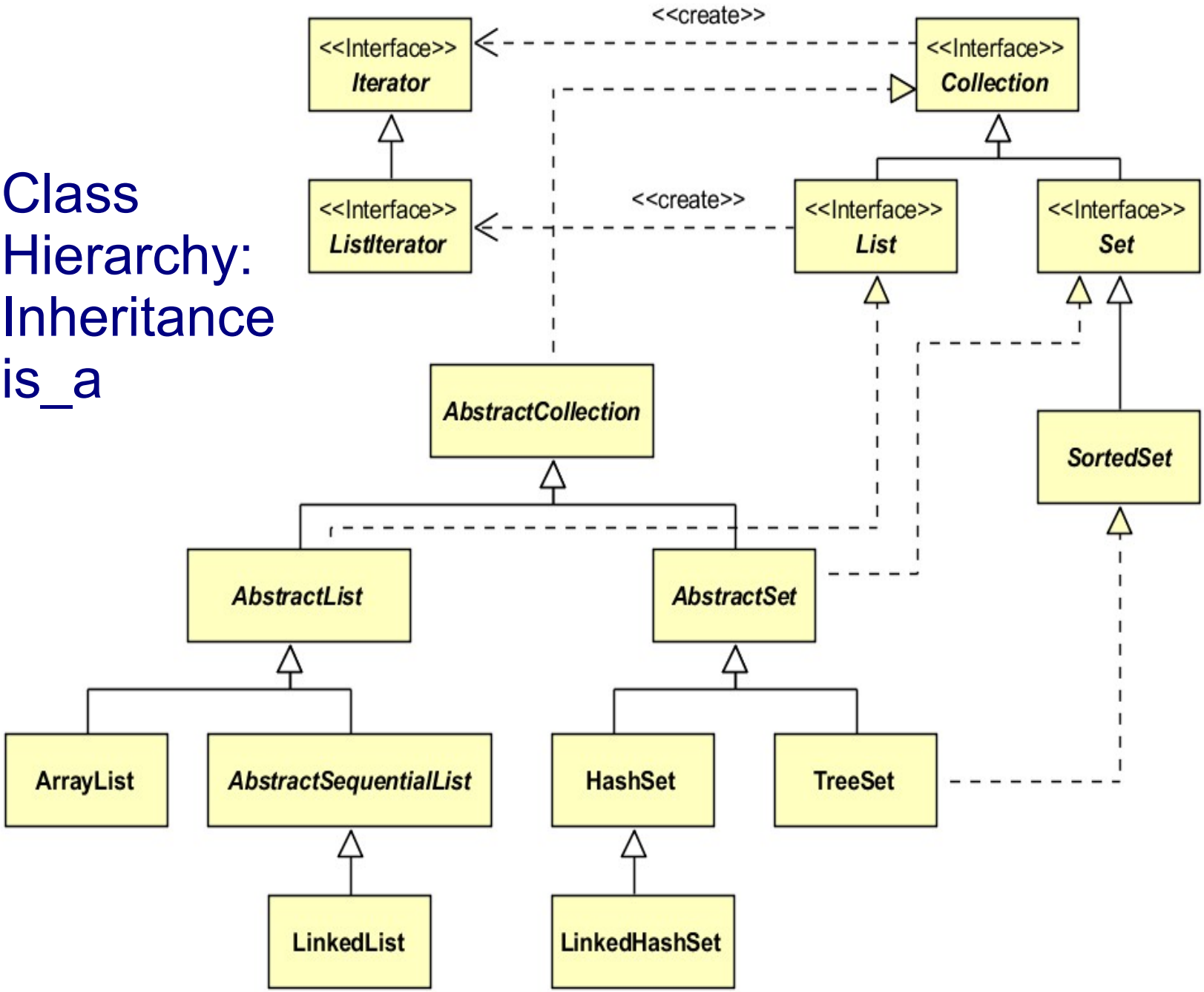
– declaration + initialization (shorter form, applies only to the class String):

```
String s = "Hello Java World";
```

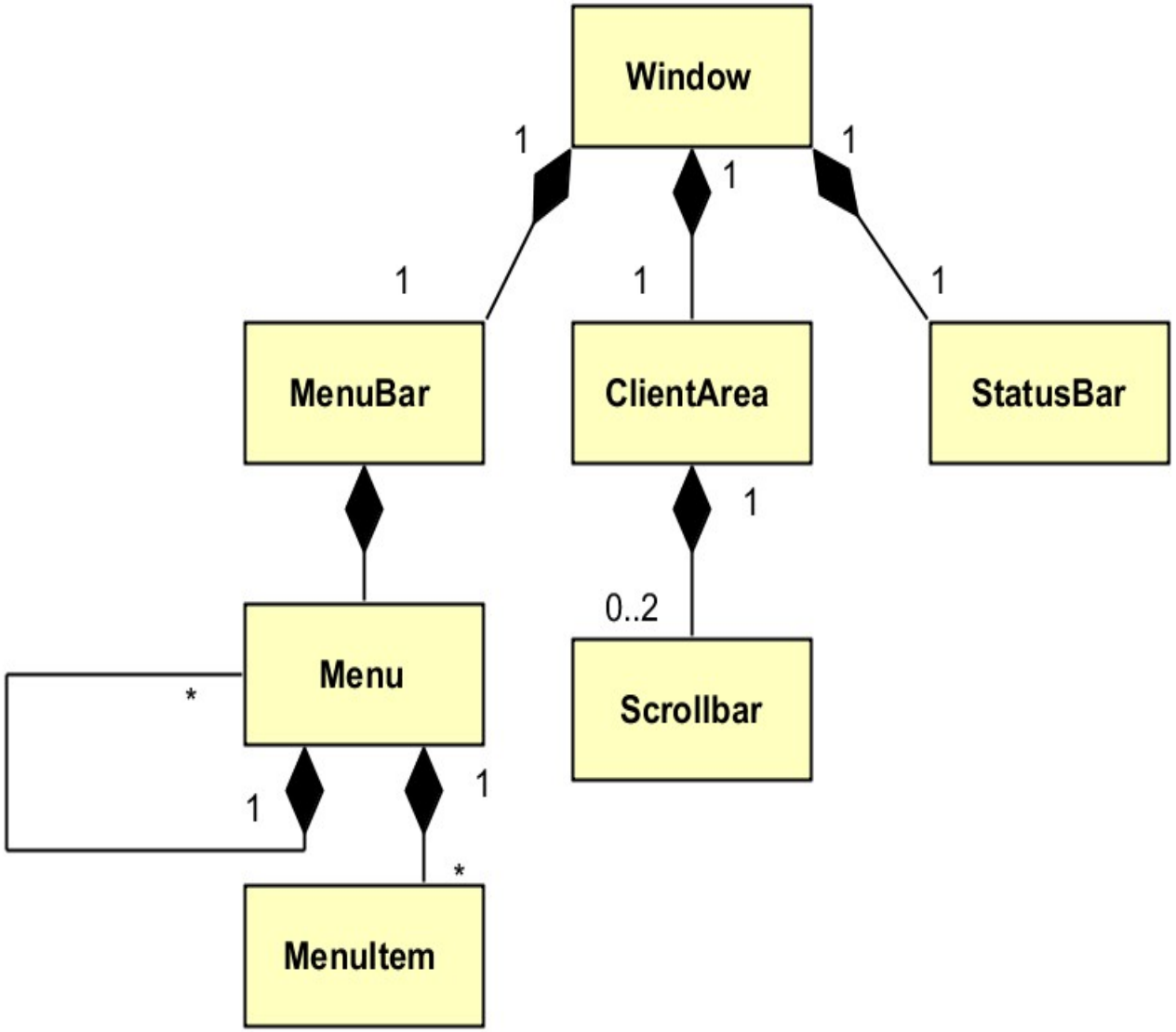
SOLID Design Principles of OOP

1. Single responsibility principle - a class should only have a single responsibility, that is, only changes to one part of the software's specification should be able to affect the specification of the class.
2. Open–closed principle - software entities should be open for extension, but closed for modification.
3. Liskov substitution principle - Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.
4. Interface segregation principle - Many client-specific interfaces are better than one general-purpose interface.
5. Dependency inversion principle - depend upon abstractions, not concretions.

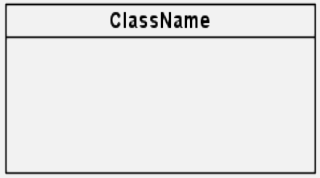
Class Hierarchy: Inheritance is_a

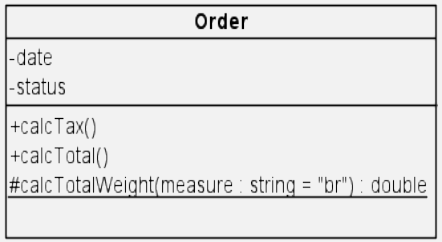


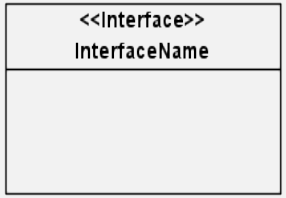
Object Hierarchy: Composition, has_a

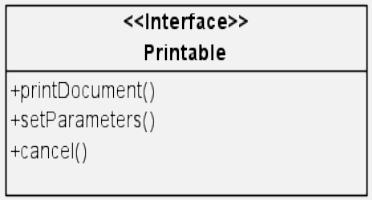


Elements of Class Diagrams





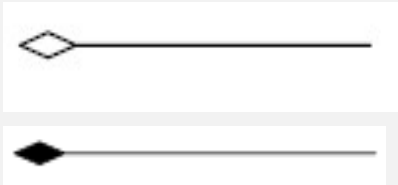




Types of connections:

- Association

- aggregation
- composition

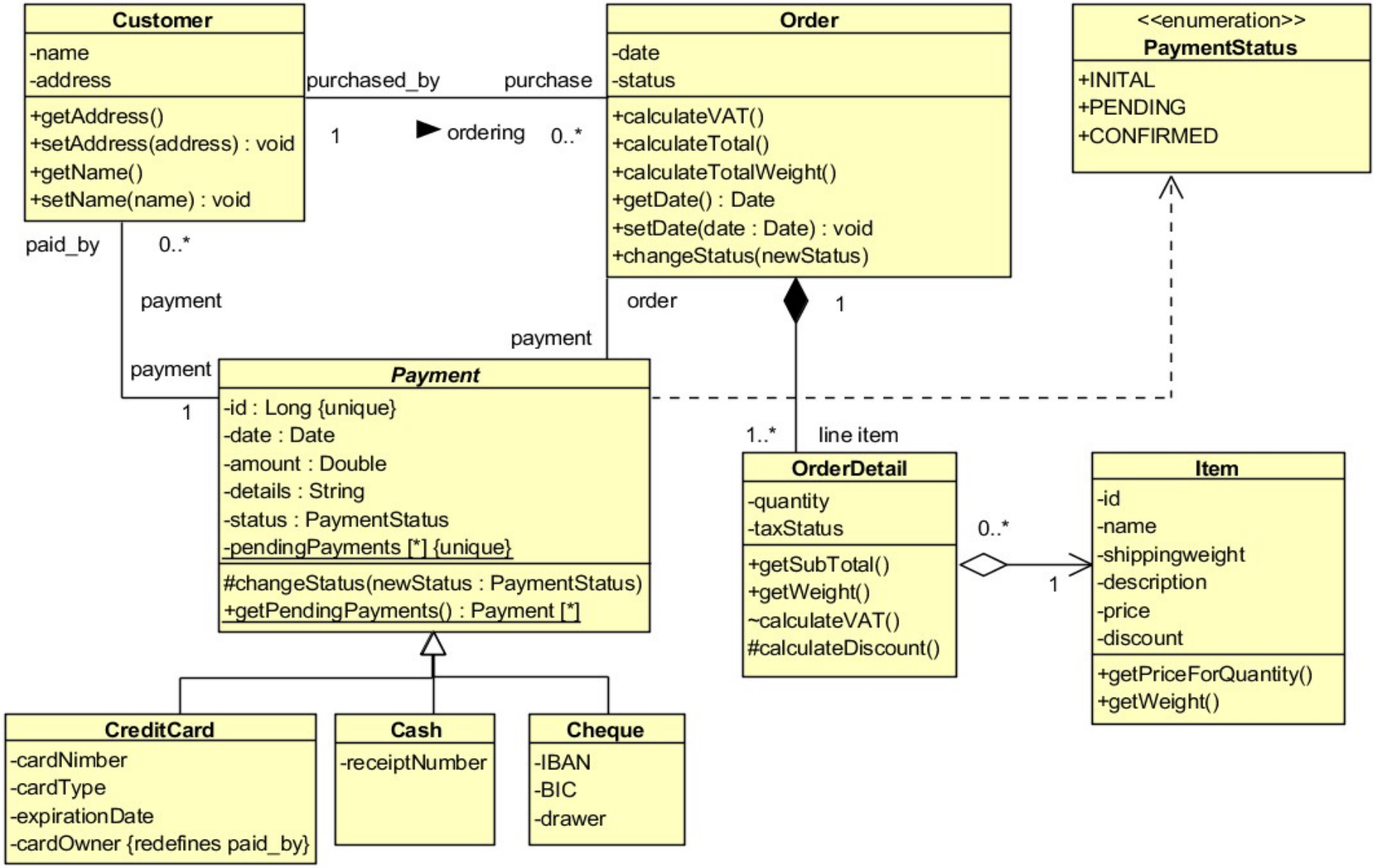


- dependence

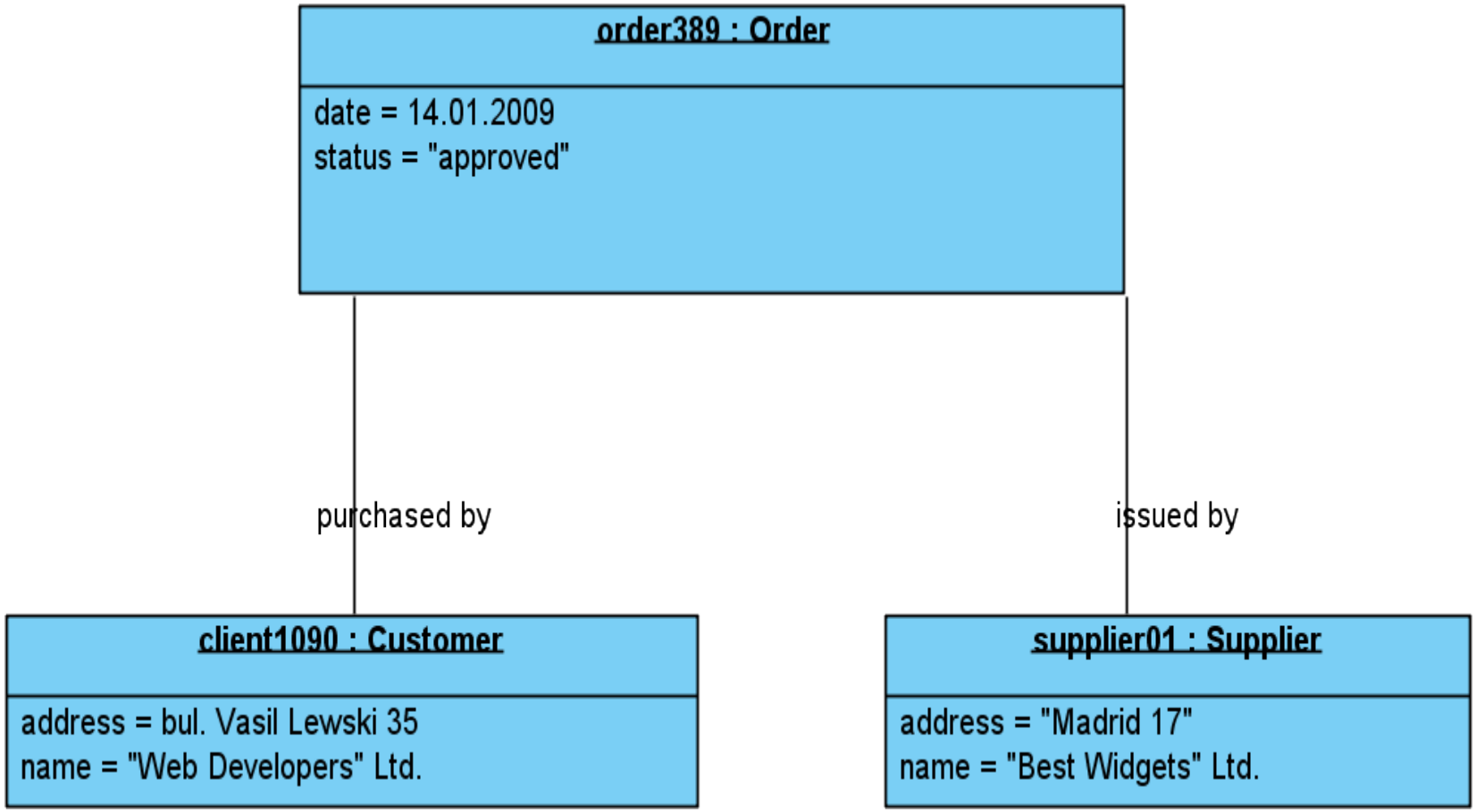
- generalization

- realization

Class Diagram



Object Diagram



Packages and Access Specifiers

- Packages and directories
- Importing packages – import
- Access specifiers
  - **public**
  - **private**
  - **protected**
  - Friendly access – by default within the package

Primitive and Object Data Types

- Primitive data types, object wrapper types and default values for attributes of primitive type
  - boolean --> Boolean — false
  - char --> Character — '\u0000'
  - byte --> Byte — (byte) 0
  - short --> Short — (short) 0
  - int --> Integer — 0
  - long --> Long — 0L
  - float --> Float — 0.0F
  - double --> Double — 0.0D
  - void --> Void

❖BigInteger and BigDecimal - higher-precision numbers

Primitive Type Literals

- in decimal notation:
  int: 145, 2147483647, -2147483648
  long: 145L, -1l, 9223372036854775807L
  float: 145F, -1f, 42E-12F, 42e12f
  double: 145D, -1d, 42E-12D, 42e12d
- in hexadecimal notation: 0x7ff, 0x7FF, 0X7ff, 0X7FF
- in octal notation: 0177
- in binary notation: 0b11100101, 0B11100101

Object (Reference) Data Types

- Initialization with default values
- Value of uninitialized reference = null
- Declaring class methods

```
class Person {
    String name;
    int age;
    String changeNameAndAge (String aName, int anAge) {
        name = aName;
        age = anAge;
        return “Name: “ + name + “Age: “ + age;
    }
}
```

Method Name

Arguments

Return Type

Method Body

Returning Value

Object Constructors in Java

- Initialization of objects with constructors
- Overloading of constructors and other methods
- Default constructors
- Reference to the current object – **this**

Objects Initialization. Array initialization

- Initialization in declaration
- Initialization in constructor
- „Lazy“ initialization
- Initialization of static class members
- One-dimensional and multi-dimensional arrays
- Array initialization

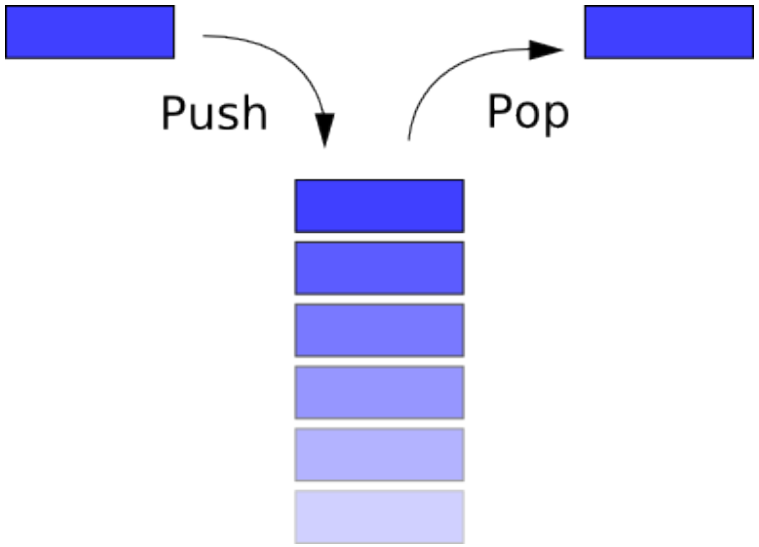
Memory Types

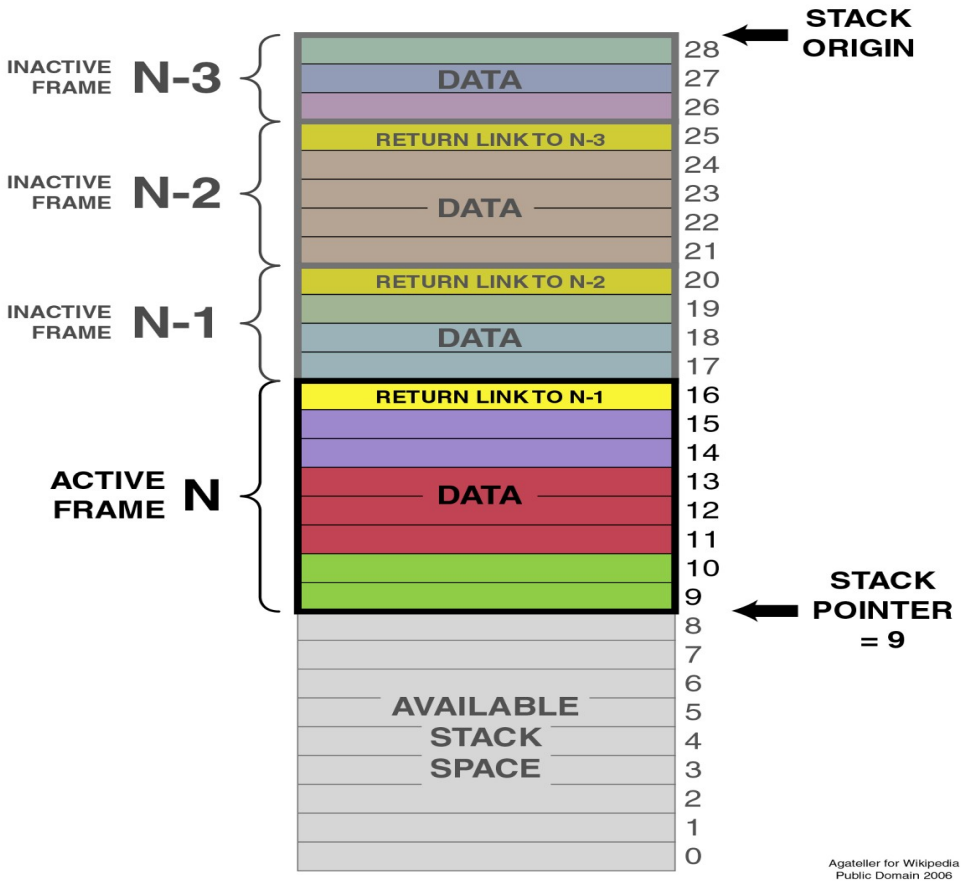
- Register memory - CPU registers, fast, small numbers stored operand instructions just before treatment
- Program Stack = Last In, First Out (LIFO) – Keep primitive data types and references to objects during program execution
- Dynamically allocated memory – Heap – can store different sized objects for different periods of time, can create new objects dynamically and to be released – Garbage Collector
  - Young generation – objects that exist for short period
  - Old generation – objects that exist longer
  - Permanent Generation = class definitions.

Java 8+ Metaspace

- Constant storage, non-RAM storage (external memory)

Program Stack





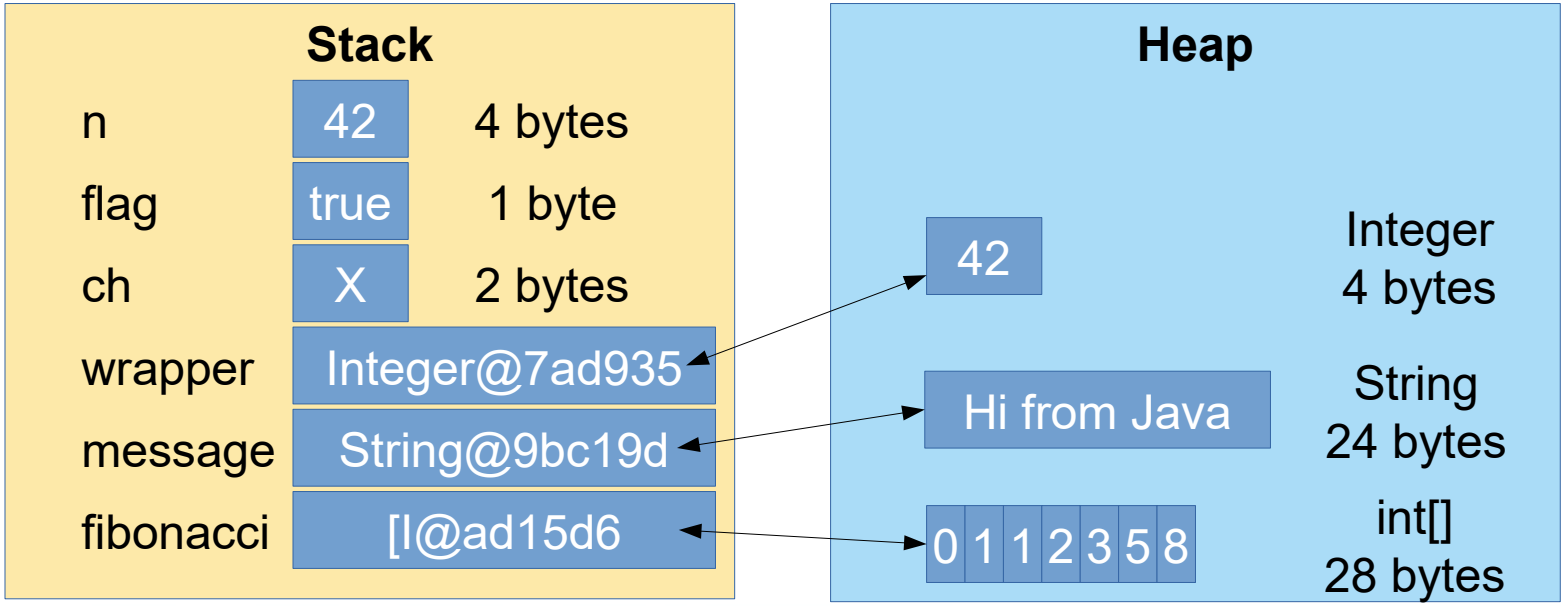
Source: http://en.wikipedia.org/wiki/File:ProgramCallStack2.png

C:\Windows\system32\cmd.exe

```
c:\CourseAdvancedJavaVerint\Temp>jstack 1612
2015-07-16 15:52:18
Full thread dump Java HotSpot(TM) 64-Bit Server VM (25.45-b02 mixed mode):

"DestroyJavaVM" #21 prio=5 os_prio=0 tid=0x00000000024b8000 nid=0x1f04 waiting on condition [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE

"Thread-9" #20 prio=5 os_prio=0 tid=0x000000000bea7000 nid=0x2348 waiting for monitor entry [0x000000000d14f000]
   java.lang.Thread.State: BLOCKED (on object monitor)
        at simpletest.TwoThreadsSynchronizedCounter.lambda$0(TwoThreadsSynchronizedCounter.java:14)
        - waiting to lock <0x00000000d5e660a0> (a java.lang.Object)
        at simpletest.TwoThreadsSynchronizedCounter$$Lambda$1/424058530.run(Unknown Source)
        at java.lang.Thread.run(Thread.java:745)

"Thread-8" #19 prio=5 os_prio=0 tid=0x000000000bea5800 nid=0x6ac waiting for monitor entry [0x000000000ca2e000]
   java.lang.Thread.State: BLOCKED (on object monitor)
        at simpletest.TwoThreadsSynchronizedCounter.lambda$0(TwoThreadsSynchronizedCounter.java:14)
        - waiting to lock <0x00000000d5e660a0> (a java.lang.Object)
        at simpletest.TwoThreadsSynchronizedCounter$$Lambda$1/424058530.run(Unknown Source)
        at java.lang.Thread.run(Thread.java:745)

"Thread-7" #18 prio=5 os_prio=0 tid=0x000000000bea5000 nid=0x1ffc waiting for monitor entry [0x000000000cfcf000]
   java.lang.Thread.State: BLOCKED (on object monitor)
        at simpletest.TwoThreadsSynchronizedCounter.lambda$0(TwoThreadsSynchronizedCounter.java:14)
        - waiting to lock <0x00000000d5e660a0> (a java.lang.Object)
        at simpletest.TwoThreadsSynchronizedCounter$$Lambda$1/424058530.run(Unknown Source)
        at java.lang.Thread.run(Thread.java:745)

"Thread-6" #17 prio=5 os_prio=0 tid=0x000000000bea2000 nid=0x40c waiting for monitor entry [0x000000000cd5f000]
   java.lang.Thread.State: BLOCKED (on object monitor)
        at simpletest.TwoThreadsSynchronizedCounter.lambda$0(TwoThreadsSynchronizedCounter.java:14)
        - waiting to lock <0x00000000d5e660a0> (a java.lang.Object)
        at simpletest.TwoThreadsSynchronizedCounter$$Lambda$1/424058530.run(Unknown Source)
        at java.lang.Thread.run(Thread.java:745)

"Thread-5" #16 prio=5 os_prio=0 tid=0x000000000bea0800 nid=0x1708 waiting for monitor entry [0x000000000ceae000]
   java.lang.Thread.State: BLOCKED (on object monitor)
        at simpletest.TwoThreadsSynchronizedCounter.lambda$0(TwoThreadsSynchronizedCounter.java:14)
        - waiting to lock <0x00000000d5e660a0> (a java.lang.Object)
        at simpletest.TwoThreadsSynchronizedCounter$$Lambda$1/424058530.run(Unknown Source)
        at java.lang.Thread.run(Thread.java:745)

"Thread-4" #15 prio=5 os_prio=0 tid=0x000000000be9d000 nid=0xc0c waiting for monitor entry [0x000000000c7df000]
   java.lang.Thread.State: BLOCKED (on object monitor)
        at simpletest.TwoThreadsSynchronizedCounter.lambda$0(TwoThreadsSynchronizedCounter.java:14)
        - waiting to lock <0x00000000d5e660a0> (a java.lang.Object)
        at simpletest.TwoThreadsSynchronizedCounter$$Lambda$1/424058530.run(Unknown Source)
        at java.lang.Thread.run(Thread.java:745)

"Thread-3" #14 prio=5 os_prio=0 tid=0x000000000be9c800 nid=0x2394 waiting for monitor entry [0x000000000cc2f000]
   java.lang.Thread.State: BLOCKED (on object monitor)
```

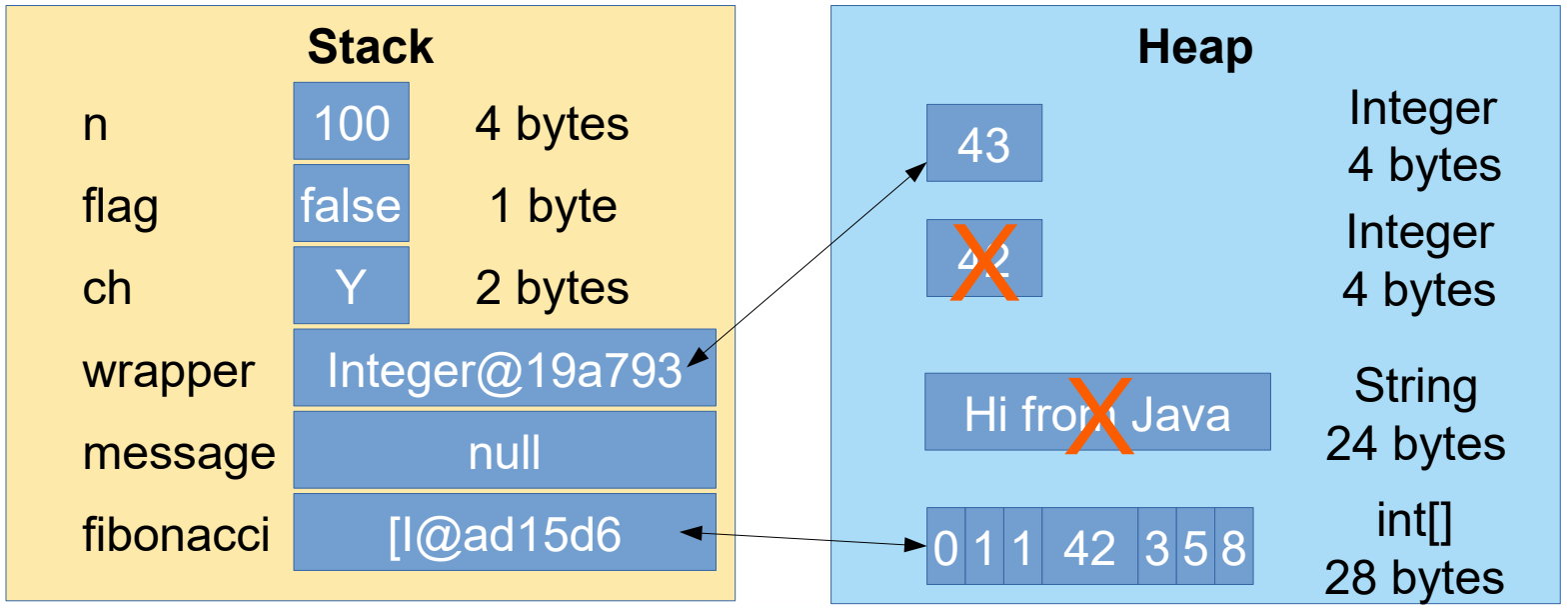
Stack and Heap (Quick Review)

```
int n = 42;
boolean flag = true;
char ch = 'X';
Integer wrapper = n;
String message = "Hi from Java!";
int[] fibonacci = { 0, 1, 1, 2, 3, 5, 8 };
```



Stack and Heap (Quick Review)

```
n = 100;
flag = !flag;
h = ++ch;
wrapper = ++wrapper;
message = null;
fibonacci[3] = 42;
```



Variable Scopes

```
public class VarScopes {
    static int s1 = 25;
    int i1 = 350;
    public static void main(String[] args) {
      if(s1 > 10){
          int a = 42;
          // Only a available
          {
              int b = 108;// Both a & b are available
          }
          // Only a available, b is out of scope
      }
      // a & b are out of scope
   }
}
```

Operators in Java - I

- Assignment operator
- Mathematical operators
- Relational operators
- Logical operators
- Bitwise operators
- String operators
- Operators for type conversion
- Priorities of operators

Operators in Java - II

- Each operator has priority and associativity - for example, + and – have a lower priority from * and /
- The priority can be set clearly using brackets (and) - for example (y – 1) / (2 + x)
- According associativity operators are left-associative, right-associative and non-associative: For example:
  x + y + z => (x + y) + z, because the operator + is left-associative
- if it was right associative, the result would be x + (y + z)

Operators in Java - III

- Assignment operator: **=**
  - is not symmetrical – i.e. **x = 42** is OK, **42 = x** is NOT
  - to the left always stands a variable of a certain type, and to the right an expression from the same type or type, which can be automatically converted to present

- Mathematical operators:
  - with one argument (unary): **-, ++, --**
  - with two arguments (binary): **+, -, *, /, %** (remainder)

- Combined: **+=, -=, *=, /=, %=**

For example: **a += 2** <=> **a = a + 2**

Send Arguments by Reference or by Value

- Formal and actual arguments - Example:

Static method - no **this**

Formal Argument
- copies the actual value

```
public static void incrementAgeBy10(Person p){
    p.age = p.age + 10;
}

Person p2 = new Person(23434345435L, "Petar
Georgiev", "Plovdiv", 39);
incrementAgeBy10(p2);
System.out.println(p2);
```

Actual Argument

Send Arguments by Reference and Value

- Case A: When the argument is a primitive type, the formal argument copies the actual value
- Case B: When the argument is a object type, the formal argument copies reference to the actual value
- Cases A & B: Changes in the copy (formal argument) does not reflect the actual argument
- However, if formal and actual argument point to the same object (Case B) – then **changes in properties (attribute values) of this object are available from the calling method** – i.e. we can return value from this argument

Operators in Java - IV

- Relational operators (comparison): **==, !=, <=, >=**
- Logical operators: **&&** (AND), **||** (OR) and **!** (NOT)

the expression is calculated from left to right only when it's necessary for determining the final outcome

- Bitwise operators: **&** (AND), **|** (OR) and **~** (NOT), **^** (XOR), **&=**, **|=**, **^=**
- Bitwise shift: **<<**, **>>** (preserves character), **>>>** (always inserts ziros left – does not preserve character), **<<=**, **>>=**, **>>>=**

Operators in Java - V

- Triple **if-then-else** operator:

<boolean-expr> ? <then-value> : <else-value>

- String concatenation operator: **+**

- Operators for type conversion (type casting):

(byte), **(short)**, **(char)**, **(int)**, **(long)**, **(float)** ...

- Priorities of operators:

unary > binary arithmetical > relational > logical > three-argumentative operator **if-then-else** > operators to assign a value

Controlling Program Flow - I

- Conditional operator - **if-else**
- Returning Value – **return**
- Operators organizing cycle - **while, do while, for, break, continue**
- Operator to select one from many options - **switch**

Controlling Program Flow - II

- Conditional operator **if-else**:

if(<boolean-expr>)

<then-statement>

or

if(<boolean-expr>)

<then-statement>

else

<else-statement>

Controlling Program Flow - III

- Returning value to exit the method: **return;** or
  return <value>;
- Operator to organize cycle **while**:

```
while(<boolean-expr>)
    <body-statement>
```

- Operator to organize cycle **do-while**:

```
do <body-statement>
while(<boolean-expr>);
```

Controlling Program Flow - IV

- Operator to organize cycle **for**:

for(<initialization>; <boolean-expr>; <step>)
<body-statement>

- Operator to organize cycle **foreach**:

for(<value-type> x : <collection-of-values>)
<body-statement-using-x>

Ex.: **for(Point p : pointsArray)**
System.out.println("(" +p.x + ", " + p.y + ")");

Controlling Program Flow - V

- Operators to exit block (cycle) **break** and to exit iteration cycle **continue**:

```
<loop-iteration> {
    //do some work
    continue;  // goes directly to next loop iteration
   //do more work
   break;  // leaves the loop
   //do more work
}
```

Controlling Program Flow - VI

- Use of labels with **break** and **continue**:

```
outer_label:
<outer-loop> {
    <inner-loop> {
        //do some work
        continue; // continues inner-loop
        //do more work
        break outer_label; // breaks outer-loop
        //do more work
        continue outer_label;  // continues outer-loop
    }
}
```

Controlling Program Flow - VII

❖Selecting one of several options **switch**:

```
switch(<selector-expr>) {
  case <value1> : <statement1>; break;
  case <value2> : <statement2>; break;
  case <value3> : <statement3>; break;
  case <value4> : <statement4>; break;
  // more cases here ...
  default: <default-statement>;
}
```

Enumeration Types

```
public class MyEnumeration {
   public enum InvoiceType { SIMPLE, VAT }
   public static void main(String[] args) {
     for(InvoiceType it : InvoiceType.values())
         System.out.println(it);
     }
}
```

Резултат: SIMPLE
VAT

Низове

- Класът **String** предоставя **immutable** обекти – т.е. всяка операция върху низа създава нов обект в хипа
- **StringBulider** – предоставя ефикасен откъм ресурси начин да модифициране на низове, като реализира Reusable Design Pattern: **Builder** – за постъпково изграждане на низа (основно с метод **append** и **insert**)
- Основни операции в класа **String**. Форматиран изход – метод **format()** и клас **Formatter**. Спецификатори:

%[argument_index$][flags][width][.precision]conversion

Конверсия на типа при форматиране

- d – decimal, интегрални типове
- c – character (unicode)
- b - boolean
- s - String
- f – float, double (с десетична точка)
- e - float, double (scientific notation)
- x – шестнайсетична стойност на интегрални типове
- h – шестнайсетичен хеш код

Регулярни изрази (1)

- Символни класове
  - **.** Any character (may or may not match line terminators)
  - **\d** A digit: [0-9]
  - **\D** A non-digit: [^0-9]
  - **\s** A whitespace character: [\t\n\x0B\f\r]
  - **\S** A non-whitespace character: [^\s]
  - **\w** A word character: [a-zA-Z_0-9]
  - **\W** A non-word character: [^\w]

Регулярни изрази (2)

- Квалификатори:
  - **X?** X, once or not at all
  - **X*** X, zero or more times
  - **X+** X, one or more times
  - **X{n}** X, exactly n times
  - **X{n,}** X, at least n times
  - **X{n,m}** X, at least n but not more than m times
- **Greedy, Reluctant (?) & Possessive (+)** квалификатори
- **Capturing Group - (X)**

Регулярни изрази (3)

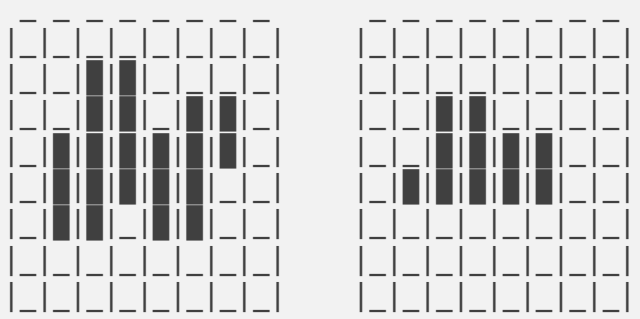
- Клас **Pattern** – основни методи:
  - **public static Pattern compile(String regex)**
  - **public Matcher matcher(CharSequence input)**
  - **public static boolean matches(String regex, CharSequence input)**
  - **public String[] split(CharSequence input, int limit)**
- Клас **Matcher** – основни методи:
  - **public boolean matches()**
  - **public boolean lookingAt()**
  - **public boolean find(int start)**
  - **public int groupCount()** и **public String group(int group)**

Problem: Word Counting

Реализирайте конзолно приложение което по подадено като аргумент от команден ред име на файл извлича top 20 ключови думи за файла на база на честотата на тяхното срещане.

Problem: Melting Iceberg

Айсберг има форма, която може да се изобрази в таблица с N реда и N стълба, 7 < N < 200, например айсбергът от фиг. 1 след един час в резултат на топенето се превръща в айсберга от фиг. 2:



фиг. 1 фиг. 2

Клетките от първия и последния ред и стълб са винаги празни. Външните клетки, които са изложени на съприкосновение с топлия въздух и вода се топят, а вътрешните не. Айсбергът се топи по следното правило: всяка клетка която има поне 2 от съседните 4 клетки (с обща страна) празни се стопява изцяло за 1 час, а останалите клетки не се топят изобщо. Напишете програма, която прочита от текстов файл размера и съдържанието на таблицата:

```
8
00000000
00**0000
00**0**0
0******0
0*****00
0**0**00
00000000
00000000
```

В резултат програмата следва да извежда на екрана броя часове, за които айсбергът ще се разтопи изцяло. В горния пример изходът на програмата следва да бъде: 4.

Problem: Melting Iceberg II

Реализирайте конзолно приложение за интерактивно въвеждане и редактиране на таблицата от задача 3. Приложението следва да поддържа текстово меню с възможности за:

1) редактиране на таблицата;

2) създаване на нова празна таблица с възможност за редактиране;

3) прочитане на таблицата от текстов файл;

4) запис на таблицата в текстов файл;

5) изход от програмата.

Редактирането на таблицата трябва да стане в текстов вид, интерактивно от клавиатурата с поддържане на активен курсор (символ '#') и с натискане на '+' за запълване на клетката където е курсора и '-' за изчистване на клетката, където е курсора. Преместването на курсора става със стрелките от клавиатурата.

След натискане на всеки клавиш се извежда цялата таблица и един празен ред за разделител. Редактирането приключва с натискане на клавиша <Enter>, след което се връщаме в главното меню на програмата, като редактираната таблица се запомня.

Thank's for Your Attention!

Trayan Iliev

CEO of IPT – Intellectual Products & Technologies

http://iproduct.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD