Overview of Java

Tutorial:
https://enos.itcollege.ee/~japoia/docs/tutorial/

Textbook:
https://math.hws.edu/javanotes-swing/

Java – History

- 1991 - P. Naughton, J. Gosling: project "Green", Sun virtual machine, Oak ==> **Java**
- 1992 - "*7" - video-, cableTV equipment
- 1993 - Mosaic ==> HotJava web browser (P.Naughton, J.Payne)

- 1996 - JDK 1.02 Java
- 1997 – JDK 1.1 AWT, UniCode
- 1998 - Java 2 (JDK 1.2..) Swing
- 1999 – J2EE Collections API
- 2004 – Java 5 Generic data types, autoboxing

- 2014 – Java 8 (LTS) lambdas, streams, default methods
- 2017 – Java 9 modules, linking, JShell
- 2018 – Java 10 var-types
- 2018 – Java 11 (LTS) var-types inside lambdas
- 2019 – Java 12 switch-expressions
- 2019 – Java 13 new „->“-switch statement, text blocks in triple-quotes
- 2020 – Java 14 records
- 2021 – Java 17 (LTS) hidden and weak classes
- 2023 – Java 21 (LTS) sealed classes/interfaces
- 2024 – Java 23 foreign functions (e.g. C) / memory API
- 2025 – Java 24 / 25

https://en.wikipedia.org/wiki/Java_version_history

Version	Release Date	Support End (LTS)	Key Language-Level Features
Java 5	Sept 2004	Oct 2009	**Generics**, Annotations, Enums, Autoboxing/Unboxing, Varargs, Enhanced for loop (for-each).
Java 8	March 2014	Dec 2030*	**Lambda Expressions**, **Stream API**, Optional, Default Methods in Interfaces, New Date/Time API.
Java 11	Sept 2018	Sept 2026	**Local-Variable Syntax for Lambda Parameters** (var), HTTP Client API (Standard), Flight Recorder.
Java 17	Sept 2021	Sept 2029	**Sealed Classes**, **Records**, Pattern Matching for instanceof, Text Blocks, Switch Expressions.
Java 21	Sept 2023	Sept 2031	**Virtual Threads** (Project Loom), Record Patterns, Pattern Matching for switch, Sequenced Collections.
Java 25	Sept 2025	Sept 2033	String Templates (Standardized), Flexible Constructor Bodies, Implicitly Declared Classes.

Features

- C-like syntax, simpler than C++ (more similar to C#)
- No preprocessor
- Object oriented (single inheritance until Java 7, automatic garbage collection, late binding, abstract classes and interfaces)
- Standard and rich APIs (graphics, I/O, data structures, networking, multithreading, …)
- Standard documentation format and documenting tools

- Supported by variety of development environments (IDEs): IntelliJ, Eclipse, NetBeans, …
- Mostly interpretive language, binary representation of a program is platform independent Java bytecode interpreted by Java Virtual Machine (JVM)
- JIT (just-in-time) compilation to machine code
- JVM is used by many other languages (Scala, Clojure, Groovy, Kotlin, ...)

Drawbacks

- Low-level (hardware) programming is hard
- Interpretive => slow (bias); not true anymore
- Not flexible enough to create totally new abstractions; better starting from Java 8
- Not suited for beginners, simple programs look complex, steep learning curve, huge ecosystem

General Structure

- Platform API = “technology” (J2SE, J2EE, J2ME, JFX, JavaCard, Java DB, Java TV…)
  Java SE – Java Standard Edition, JDK 17 (JRE included)
- Modules (>Java 9): java.base, javafx.base, javafx…., …
- Packages (flat namespace): java.lang, java.util, java.io, java.nio, java.net, java.math, …
  Default package is unnamed, package java.lang is always present, other packages need to be imported
- Classes, interfaces, abstract classes – hierarchy (single inheritance for classes, multiple interfaces allowed)
  Class Object is the root of class tree and default parent class if „extends“ clause is missing

Structure of the Class

Data

- Class variables (static), common for all objects, e.g. constants (final)
- Instance variables, individual data, also known as “attributes” or “fields” or “properties”
- Possible inner classes, …

Actions

- Class methods (static), imperative paradigm
  - Constructors to create new objects
- Instance methods (work on objects)

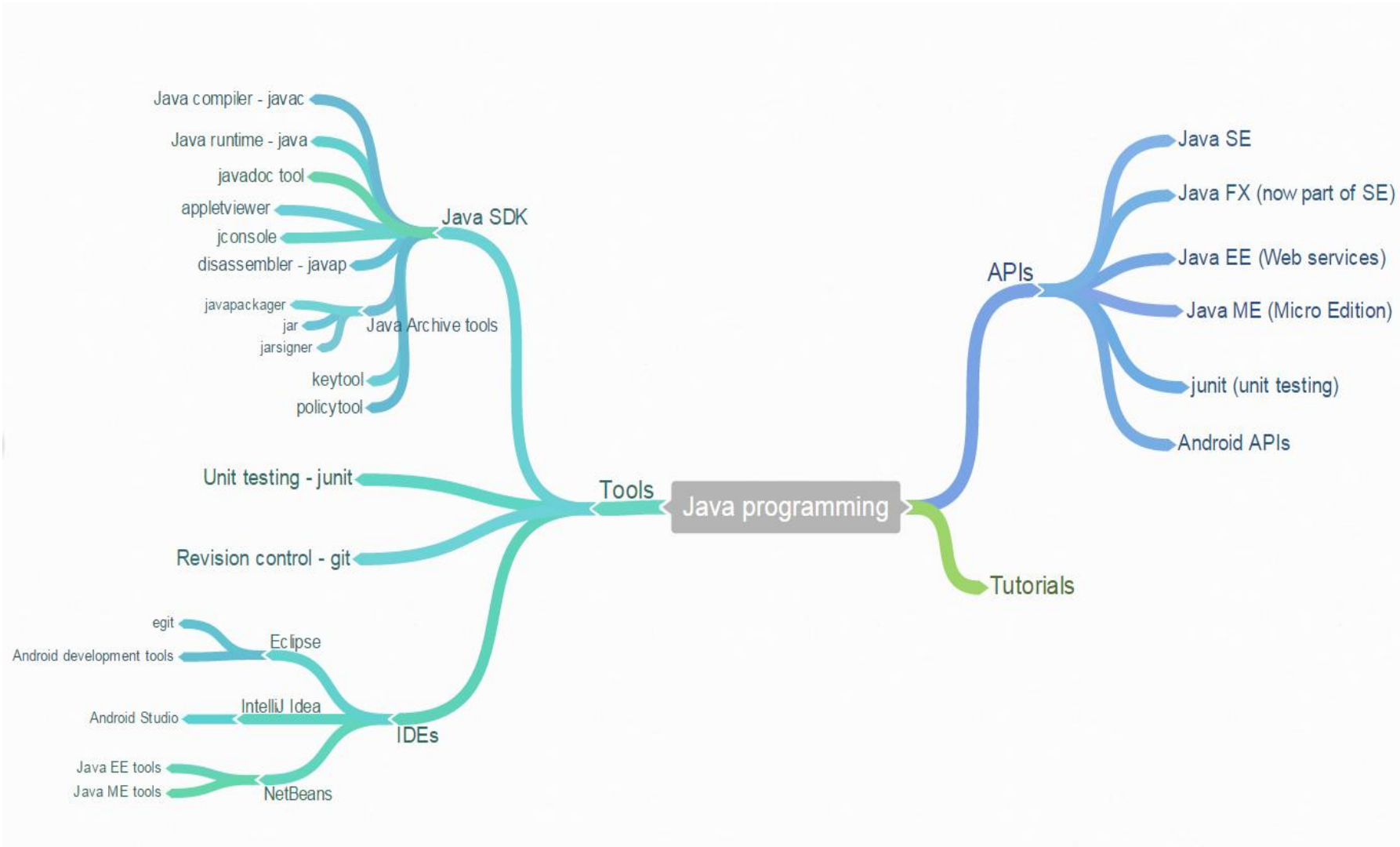
JDK – Java Development Kit

`javac` – Java compiler, X.java => X.class

`java` - Java interpreter, executes X.class

`javadoc` – documentation generator, generates *.html from javadoc comments in program source

…


Java programming
Tools
Java SDK
Java compiler - javac
Java runtime - java
javadoc tool
appletviewer
jconsole
disassembler - javap
Java Archive tools
javapackager
jar
jarsigner
keytool
policytool
Unit testing - junit
Revision control - git
IDEs
Eclipse
egit
Android development tools
IntelliJ Idea
Android Studio
NetBeans
Java EE tools
Java ME tools
APIs
Java SE
Java FX (now part of SE)
Java EE (Web services)
Java ME (Micro Edition)
junit (unit testing)
Android APIs
Tutorials

Examples

Life cycle of a program:

edit, compile, debug syntax, …, run, debug semantics, …, run, test, …, test …

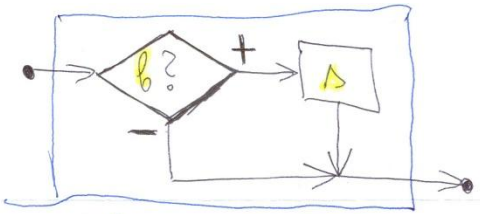
- `First.java`
- `demo/Example.java`
- `Control.java`

Statements (as in Java 8)

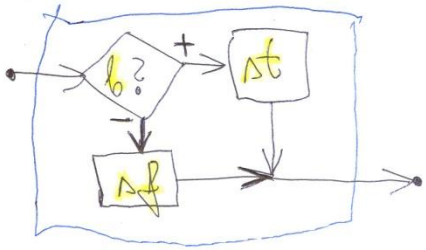
- block `{ declarations; statements }`
- expression
  - method call: `String.valueOf (56); s.length();`
  - constructor call: `new StringBuilder();`
  - assignment: `variable = expression`
  - complex expression containing operators: `a+b*(c-d)`
- empty statement and labelled statement
- if statement and if-else statement
- switch statement (new approach in Java 12 and 13)

- for statement
- while statement and do-while statement
- break statement
- continue statement
- return statement
- throw statement
- try-catch construction (try statement)
- `synchronized (object) block;` (synchronized statement)
- assert statement

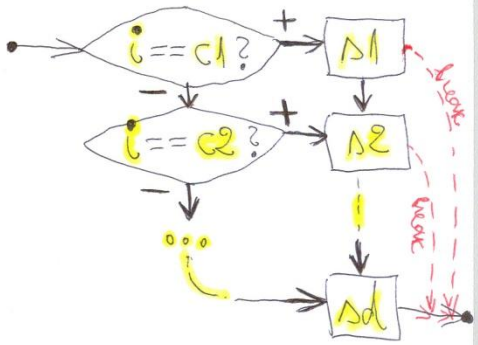
```
if (b) s;
```



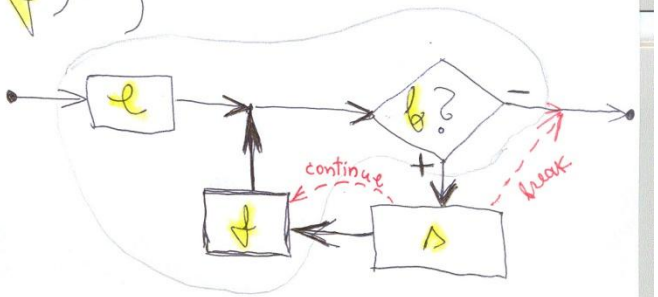

```
if (b) st; else sf;
```



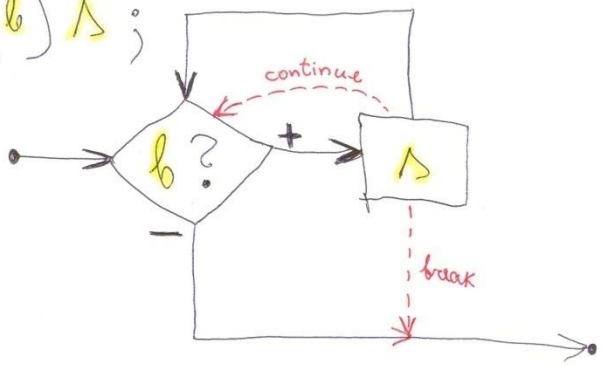

```
switch (i) {
    case c1: s1;
    case c2: s2;
    ...
    default: sd;
}
```



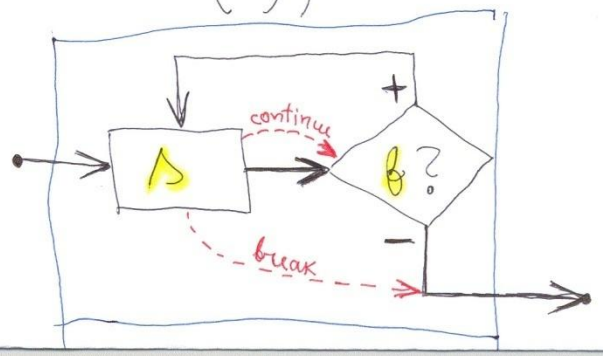

```
for (e; b; f) s;
```



while (b) A ;



do A while (b);



Examples

- Mswitch
- Mswitchbreak

Data Structures in Java

- Simple variables: primitive types and object types
- Arrays: base type, index, length
- Objects: encapsulate different fields into one instance (like records in “old” imperative languages + methods)
- Collections – built-in tools in Java API to manipulate group of objects: Vector, Hashtable, etc. Java collections framework

Types

- Primitive types: byte, short, int, long, float, double, boolean, char
- Object types:
  - Wrappers: Byte, Short, Integer, Long, Float, Double, Boolean, Character
  - Other API types: String, Object, StringBuffer, …
  - Interface types: Comparable, Runnable, …

Arrays

Array creation and initialization

```
int [] a = { 1, 5, 8 };
```

consists of 3 steps:

```
int [] a;                    // variable declared
a = new int [3];             // memory allocated
a[0]=1; a[1]=5; a[2]=8; // values assigned
int n = a.length;            // array size
```

Array expression:

```
int [] a = new int [] { 1, 5, 8};
```

Multi-dimensional Arrays

- 2-dimensional array is an array of 1-dimensional arrays (NB! these can be of different length):

```
int [][] m;                          // 2-dim array
m = new int [2][];                   // first level
System.out.println (m.length);
m[0] = new int [4];                  // second level
m[0][0] = -8;
m[1] = new int [3];                  // different size
m[1][0] = 9;
```

Objects

Object fields:

```
class Person {
    String surname;
    String firstName;
    Calendar birthDate;
    // etc. whatever we want to record
```

Constructors

```
Person (String sn, String fn, Calendar bd) {
    surname = sn;
    firstName = fn;
    birthDate = bd;
} // constructor

Person() {
    this (“*”, “*”, Calendar.getInstance());
} // default constructor
```

Instance Methods

```
  public String toString() {
    return (firstName + " " + surname
      + " " + String.valueOf (
      birthDate.get (Calendar.YEAR)
      ) + " " + String.valueOf (
      birthDate.get (Calendar.MONTH)
      ) + " " + String.valueOf (
      birthDate.get(Calendar.DAY_OF_MONTH)));
  } // toString


} // Person
```

Usage of Objects

```
public class PersonMain {
    public static void main (String[] args) {
        Calendar bd1 = Calendar.getInstance();
        bd1.set (1959, 04, 30);
        Person p1 = new Person ("Smith",
            "John", bd1);
        System.out.println (p1);
        Person p2 = new Person();
        System.out.println (p2);
    } // main
} // PersonMain
```

Collections

Collection

- *Set* (set, unique elements)
  - HashSet
    - LinkedHashSet
  - *SortedSet* (ordered set, unique elements)
    - TreeSet
- *List* (dynamic, indexed, multiple copies allowed)
  - ArrayList
  - LinkedList
  - Vector (legacy API, similar to ArrayList)
- *Queue* (since Java 5, not discussed here)

Collections

Map ("key-value" pairs)

- HashMap
  - LinkedHashMap
- *SortedMap*
  - TreeMap
- Hashtable (legacy API)
- WeakHashMap (allow garbage collection)

Iterator (to find the next element)

Enumeration (legacy API, similar to Iterator)

Iterable (has iterator)

- *Collection*

Collections

Comparable

which of two elements is "bigger"

```
public int compareTo (Object o)
```

o1.compareTo (o2) =(/ -1, if o1 < o2
0, if o1 == o2
\ 1, if o1 > o2

Arrays

static utilities – asList, search, sort, fill, ...

Collections

static utilities – search, sort, copy, fill, replace, min, max, reverse, shuffle, ...

Java Command Line

- Javac – compiler

```
javac Cunit.java
javac -cp classpath Cunit.java
javac my/package/Myclass.java
```

- Java – interpreter

```
java Cunit any text you like to pass
java -cp classpath Cunit
java my/package/Myclass
java my.package.Myclass
```

Junit – www.junit.org

```
javac -cp .:junit-4.XX.jar ClassTest.java

java -cp .:junit-4.XX.jar org.junit.runner.JUnitCore ClassTest
```

Motivation for Object Oriented Programming

- Decrease complexity (use layers of abstraction, interfaces, modularity, ...)
- Reuse existing code, avoid duplication of code
- Support formal contracting between independent development teams
- Detect errors as early as possible (general goal of software engineering)

Motivation

- Object oriented approach was introduced on 1980-s to reduce complexity of programming large software systems (e.g. graphical user interfaces).
- Flat library of standard functions (common for early imperative programming languages) is not flexible enough to create complex software systems.
- Powerful and well organized object oriented framework makes programming easier – programmer re-uses existing codebase and specifies only these properties/functions she needs to elaborate/change (and framework adjusts to these changes).

Object

Object is characterized by

- State (defined by values of instance variables in Java)
- Behaviour (defined by instance methods in Java)
- Identity (defined by memory location in Java)

Object = Instance = Specimen = ...

- Instance variable (Java terminology) = (Object) field = Property = Attribute = ...
- Method = Subroutine = Function / Procedure = ...

Object

Encapsulation – data and operations on the data are integrated into whole (object = capsule)

ADT approach – set of operations is a part of data type

Data hiding – object state can be changed only by dedicated (usually public) methods - instance variables should be protected from direct modification

Object is an instance of the class. E.g. „Rex is a dog“.

Class

- Class defines common features of its objects („template“). E.g. „All dogs have a name“.
- Instantiation – creating a new object of the class.

- Subclass can be derived from the class – subclass inherits all the features of its parent class. Subclass allows to add new (specific) features and redefine (=override) inherited features. E.g. „Dog is (a special kind of) Animal“.
- If A is a subclass of B then B is superclass of A.

Class Hierarchy

- Generalization – common features of similar classes are described on the level of superclass (mental process – design the hierarchy of classes).

- Specialization – subclass is created to concretize (refine) certain general features and add specific data/operations to the subclass (process of coding).

Instance Methods and Class Methods

- Instance methods define the behaviour of an object (=instance).
  - `s.length()` - the length of string s in Java.

- Class methods can be used without creating an object (imperative style).
  - `Math.sqrt(2.)` - square root of 2.

  Keyword `static` in Java is used to define class methods.

Polymorphism

Same notation has different meaning in different contexts

- Two types of polymorphism:

  Overloading – operation is redefined in subclass and is binded to the activating message statically (compile time choice).

  Java constructors support overloading.

  Overriding – operation is redefined in subclass and is binded to the activating message dynamically (runtime choice).

  Java instance methods support overriding.

Examples

- Pets.java
- Phones.java
- Num.java
- Complex.java

Errors and Exceptions

Error handling without dedicated tools: return codes, global error states etc.

Problem: it is not reasonable (or even possible) to handle each unusual situation in the same place (subroutine) it occurs. How to separate error handling from the normal control flow?

Errors and Exceptions

In Java:

- try / catch (control statement)
- Throwable (specialized objects)
  - Error (program cannot continue)
  - Exception (unusual situation)
    - RuntimeException (no obligation to catch)
- throw (raise exception)
- throws (method heading - delegating)

Errors

- Error
  - LinkageError
    - ClassCircularityError
    - ClassFormatError
    - IncompatibleClassChangeError
      - NoSuchMethodError
      - NoSuchFieldError
      - InstantiationError
      - AbstractMethodError
      - IllegalAccessError
    - NoClassDefFoundError
    - VerifyError
    - AbstractMethodError
    - ExceptionInInitializationError

Errors

- ThreadDeath
- VirtualMachineError
  - InternalError
  - OutOfMemoryError
  - StackOverflowError
  - UnknownError
- AWTError

Checked Exceptions

- Exception
  - ClassNotFoundException
  - CloneNotSupportedException
  - IllegalAccessException
  - InstantiationException
  - InterruptedException
  - NoSuchMethodException
  - TooManyListenersException
  - ParseException
  - AWTException

IOException

- IOException
  - CharConversionException
  - EOFException
  - FileNotFoundException
  - InterruptedIOException
  - ObjectStreamException
    - InvalidClassException
    - InvalidObjectException
    - NotActiveException
    - NotSerializableException
    - OptionalDataException
    - StreamCorruptedException
    - WriteAbortedException

IOException

- SyncFailedException
- UnsupportedEncodingException
- UTFDataFormatException
- MalformedURLException
- ProtocolException
- SocketException
  - BindException
  - ConnectException
  - NoRouteToHostException
- UnknownHostException
- UnknownServiceException

RuntimeException

- RuntimeException
  - ArithmeticException
  - ArrayStoreException
  - ClassCastException
  - IllegalArgumentException
    - IllegalThreadStateException
    - NumberFormatException
    - FormatException
  - IllegalMonitorStateException
  - IllegalStateException
  - IndexOutOfBoundsException
    - ArrayIndexOutOfBoundsException
    - StringIndexOutOfBoundsException

RuntimeException

NegativeArraySizeException

NullPointerException

SecurityException

EmptyStackException

MissingResourceException

NoSuchElementException

IllegalComponentStateException

Try / Catch

```
try {
        block where exceptions may occur ;
}
catch (ExcType_1 variable) {
        trap_1 ;
}
...
catch (ExcType_n variable) {
        trap_n ;
}
finally {
        epilogue ;
}
```

Example

```
try {

   FileInputStream p = new FileInputStream ("/etc/passwd");
   byte[] sisu = new byte [p.available()];
   p.read (sisu);
   p.close();
   System.out.write (sisu);

} catch (FileNotFoundException e) {

    System.out.println ("File not found " + e);

} catch (IOException e) {

    System.out.println ("Input/Output error " + e);

} catch (Exception e) {

    System.out.println ("Something unusual happened " + e);

} finally {

    System.out.println (" This is finally branch");

} // try
```

Throw Statement

To raise an exception in your program

```
throw throwableObject;
```

Usually error message is provided

```
throw new SecurityException
                    ("No permission to read!");
```

All checked (not RuntimeExceptions) exceptions need handling – try/catch or delegation "up" using exception declaration in method heading

Throws Declaration

```
public static void pause()
    throws InterruptedException {
    Thread.sleep (1000);
}
public Object nextElement()
    throws java.util.NoSuchElementException {
        if (pointerToNext() == null)
            throw new
                java.util.NoSuchElementException();
        else
            return pointerToNext();
}
```

Corresponding javadoc tag!

Problems

- When extending existing exception class provide both default constructor (with no parameters) and a constructor with String parameter (error message).
- No "resume" – use loop structures to continue execution
- Declare needed variables before try-block, otherwise they are not accessible in traps (catch branches)

Examples

- Chaining – ExceptionUsage.java
- Resume and other things – Apples.java