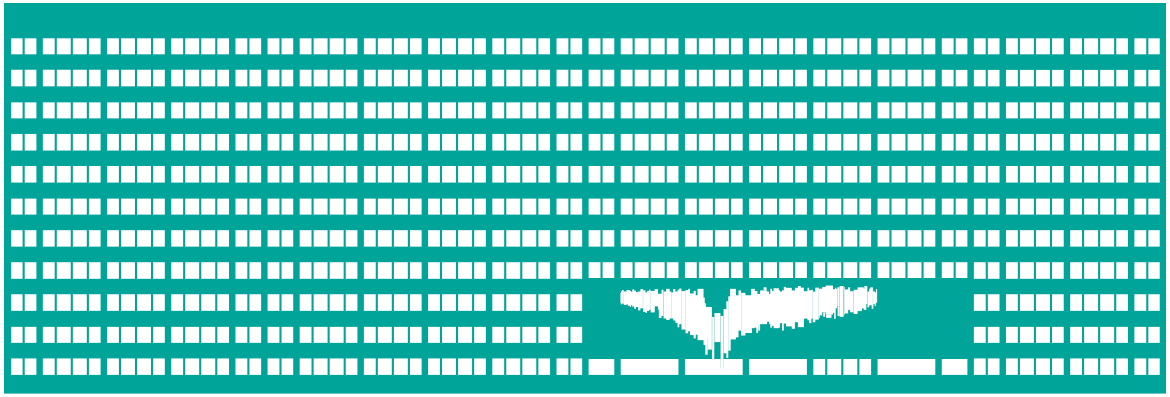
VŠB TECHNICKÁ UNIVERZITA OSTRAVA

VSB TECHNICAL UNIVERSITY OF OSTRAVA



www.vsb.cz

VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

Programming in Java 2

Jan Kožusznik

jan@kozusznik.cz

http://www.kozusznik.cz

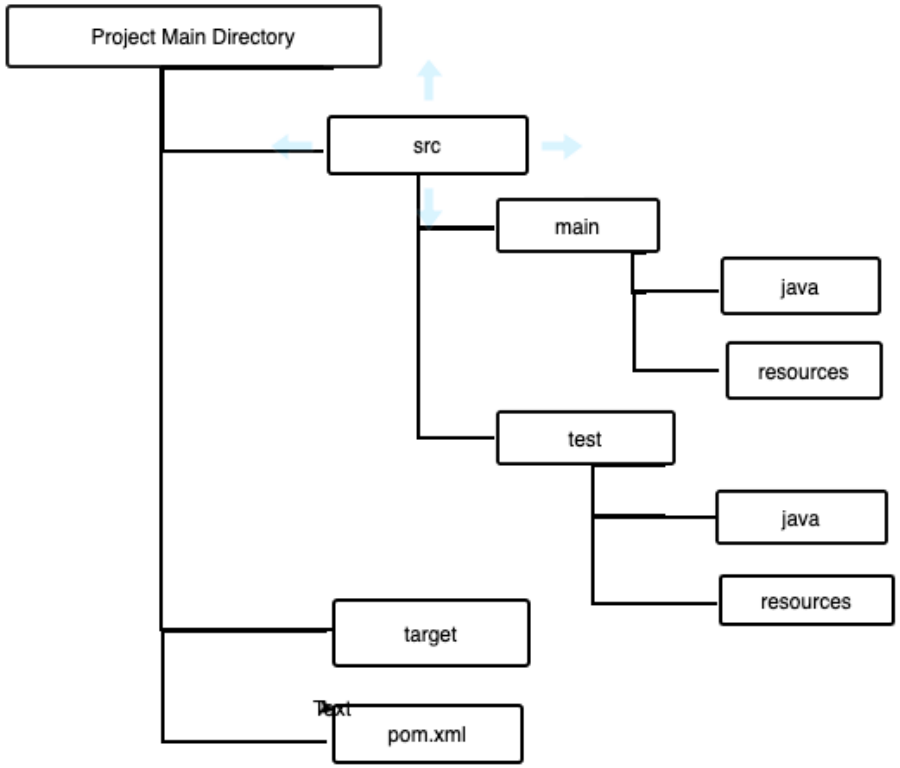
1th Lecture

- Maven
- Modules

Maven overview

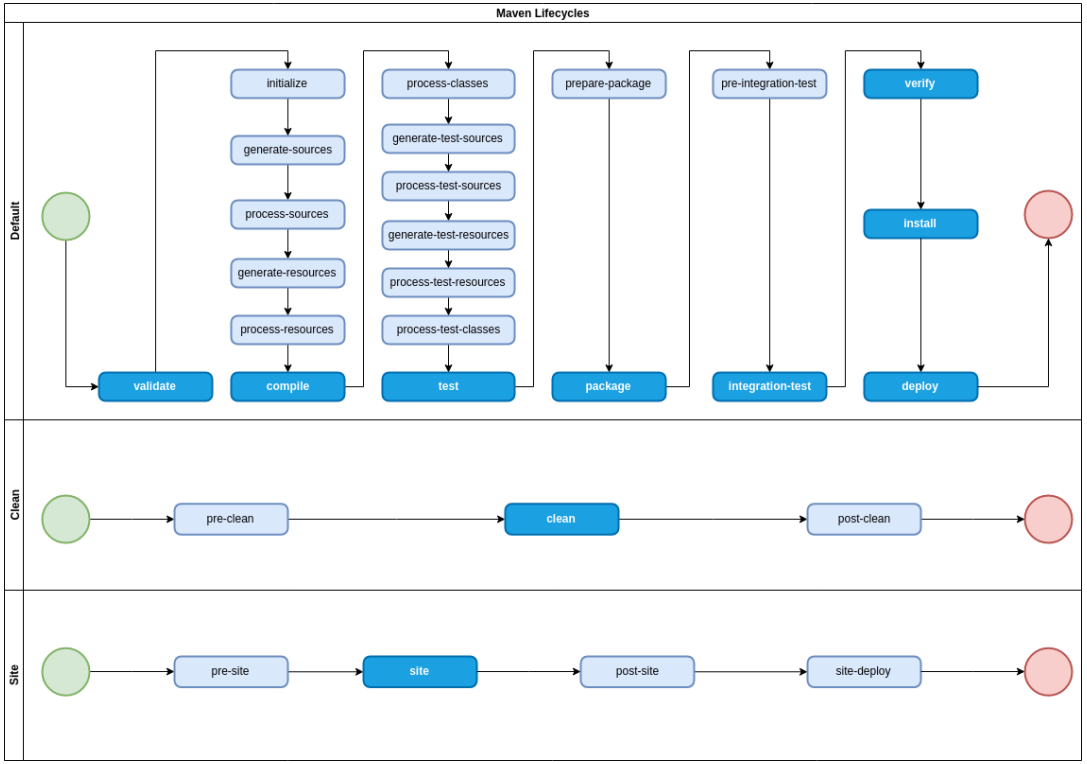
- Can generate deployable artifacts from source code
- Compile, pack, test and distribute your source code
- Manage dependencies on external libraries
- Scalable – for small projects but also for big & complex projects

Convention over configuration



Maven build lifecycles

- Three:
  - Default
  - Clean
  - Site
- Consist of phases



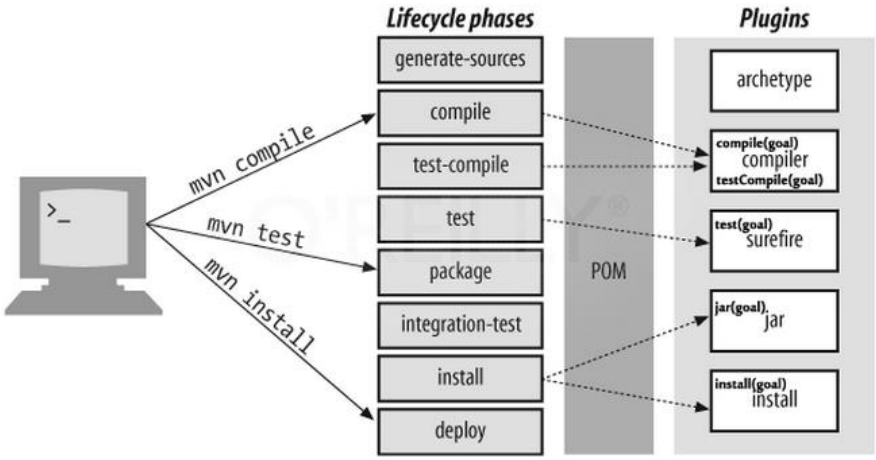
https://images.app.goo.gl/kYimcsEpSprQaREa7

Running lifecycle

```
mvn <phase>
```

- runs lifecycle containing given phase
- stops in the phase but runs every previous phase

- Provide goals
- Goals can be run .... `mvn archetype:generate`
- Goals are bound in specific phases



https://images.app.goo.gl/aJBhJZgiTkvVS2Fm8

Artifacts

- File resulting from packaging
- Definition file pom.xml
- Unique ID – artifacts coordinates:
  - artifactId
  - groupId
  - Version
- Miminal pomx.xml

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>javaII</groupId>
  <artifactId>lab01</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</project>
```

Maven Repository

- Holds build artefiсts
- Remote
  - Accessed byt http://, file://, ftp:// or other
  - Provided by a third party
  - Provided by the company to distribute private artifacts/dependencies
- Local
  - Cache of dependencies and build artifacts used or produced by your project
  - By default <HOME>/.m2/repository

Remote repository

- Can be used other then default

```
<repositories>
<repository>
<id>it4i</id>
<url>https://artifactory.cs.vsb.cz/it4i/</url>
</repository>
<repository>
<id>scijava.public</id>
<url>https://maven.scijava.org/content/groups/public</url>
</repository>
</repositories>
```

- Publish builded artifact to own repository
  - `mvn deploy`

```
<distributionManagement>
<repository>
<id>it4i</id>
<url>https://artifactory.cs.vsb.cz/it4i/</url>
</repository>
</distributionManagement>
```

Dependencies

- In pom.xml
- referenced with coordinates of the artifact
- cached in the local repository

```
<dependencies>
 <dependency>
  <groupId>org.openjfx</groupId>
  <artifactId>javafx-controls</artifactId>
  <version>15</version>
 </dependency>
 <dependency>
  <groupId>org.openjfx</groupId>
  <artifactId>javafx-fxml</artifactId>
  <version>15</version>
 </dependency>
</dependencies>
```

Dependency scope

- Compile,
- Provided,
- Runtime,
- Test,
- System,

```
<!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-api -->
<dependency>
<groupId>org.junit.jupiter</groupId>
<artifactId>junit-jupiter-api</artifactId>
<version>5.5.2</version>
<scope>test</scope>
</dependency>
```

Compile dependency

- Default for dependencies not specifying scope
- Available to all classpaths of build lifecycle (compile, test-compile, test, package)
- Packaged into final artifact

Provided dependency

- Available to compile ant test classpaths
- Not packaged as part of your artifact
- It’s expected the container, where the artifact will be used, to provide the dependency

Runtime dependency

- This dependency is not required for compiling your project
- Is required at runtime, when your application run
- Iso required when testing because tests will execute main code

Test dependency

- This dependency is only required to compile and run test
- Will not be packaged into final assembly (jar, war, ear, etc)

System dependency

- Similar to Provided Dependency
- Not looked up in repository
- Expected to exists in your development machine

Exceluded dependency

- Dependency are transitional in default
- Can by excluded

```
<dependency>
    <groupId>org.apache.velocity.tools</groupId>
    <artifactId>velocity-tools</artifactId>
    <exclusions>
        <exclusion>
            <groupId>velocity</groupId>
            <artifactId>velocity</artifactId>
        </exclusion>
        <exclusion>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
        </exclusion>
        <exclusion>
            <artifactId>commons-validator</artifactId>
            <groupId>commons-validator</groupId>
        </exclusion>
    </exclusions>
</dependency>
```

https://images.app.goo.gl/oMk8umbwHpsACokh8

Optional dependencies

- Are automatically excluded

Maven_first/pom.xml | Maven_second/pom.xml | Maven_third/pom.xml

```
        <artifactId>Maven_first</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.8</version>
        <scope>compile</scope>
        <optional>true</optional>
    </dependency>
```

https://images.app.goo.gl/VWhozoiRyYHMEAZL6

Packaging

- Determines output – jar,war, ear

```
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>javaII</groupId>
<artifactId>lab01</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>
```

Modules - goals

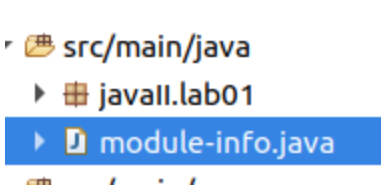
- Reliable configuration
- Strong encapsulation
- Scalable Java platform
- Greater platform integrity
- Improved performance

Modules introduction

- List modules

  java –list-modules

- Module declaration



module-info.java | lab01/pom.xml

```
1 module javaII.lab01 {
2     requires transitive javafx.controls;
3     requires javafx.fxml;
4     opens javaII.lab01 to javafx.fxml;
5     exports javaII.lab01;
6 }
```

Running java with modules

```
java --module-path <directory> --module <module>/<class with main>
```

```
java -p <directory> -m <module>/<class with main>
```

```
java -p <directory> <module>
```

requires

```
requires [transitive] [static] <module-name>
```

Export packages by modules

```
exports <package-name>

exports <package-name> to <module-name>
```

Support for services

```
uses <interface-name>

provides <interface-name> with <class-name>
…
ServiceLoader.load()
```

Allow runtime access

```
opens <package-name>

opens <package-name> to <module-name>

open module <module-name> {

}
```

Packaging as a standalone JRE

```
jlink –module-path <dirs> --add-module
<module-name> --output <out-dir>
```

```
Java --module <module-name>/<main-class>
```

Backward compatibility

- Unnamed module
- Automatic modules

Allow support for multiple java version

- META-INF
  - MANIFEST.MF … Multi-Release: true
  - Versions
    - 10
      - Com
    - 9
      - Com
- com

2nd lecture

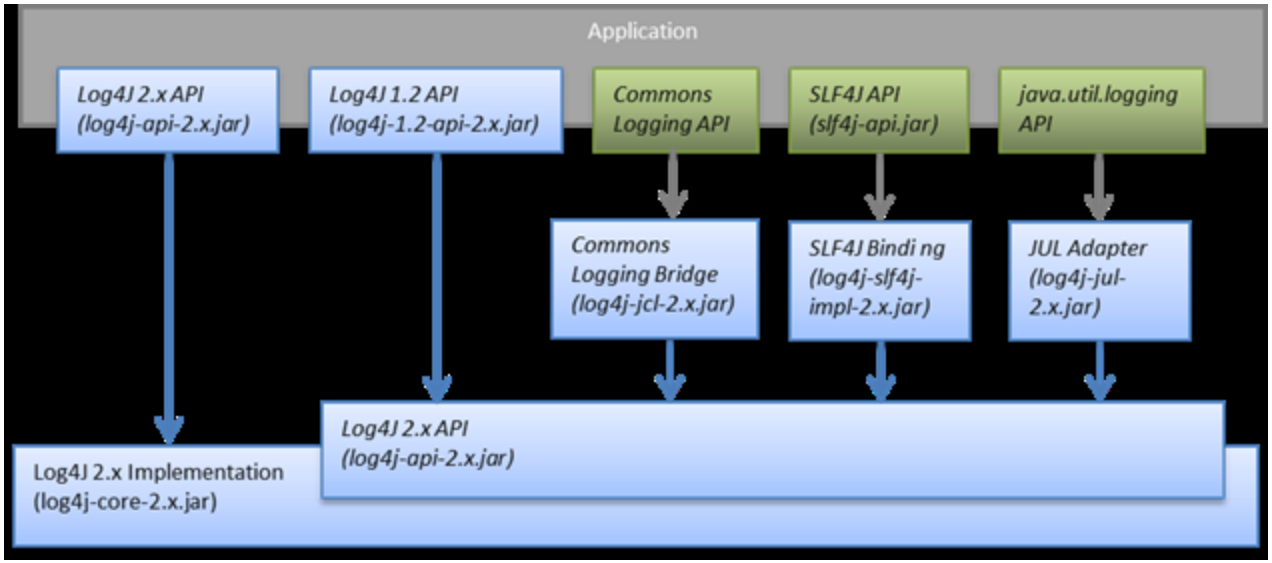
- Logging
- Assertions
- Profiling
- Effective Java:
  - Static factory methods
  - Builders
  - Correct implementation of equals

Logging

- Write runtime info with `System.out.println` is inappropriate in a production environment
- Logging framework is used instead of it (Java Util Logging, log4j, logback,log4j2,slf4j)

Log4j2

- State of The Art logging framework



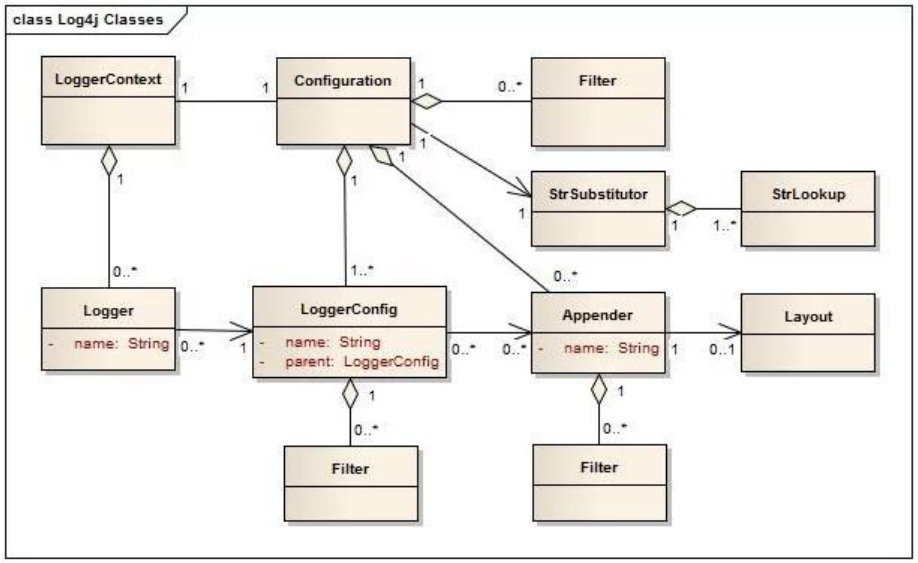
http://logging.apache.org/log4j/log4j-2.11.0/faq.html

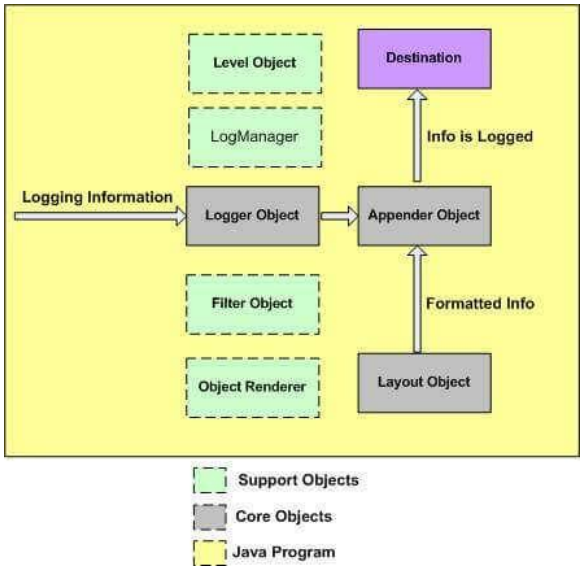
Maven dependecies

2.20.0

```
<!-- https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-api -->
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.14.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-api -->
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.14.0</version>
</dependency>
```

Log4j architecture





https://www.javatpoint.com/log4j-architecture

Logger

```
static Logger log = LogManager.getLogger(App.class);


public static void main(String[]
    log.info("Hello {}", () -> "world");
    log.log(Level.TRACE,"Hello {}",  "world");

    try {
        Files.copy(Paths.get("source"), Paths.get("path"));
    } catch (IOException e) {
        log.fatal("copying", e);
    }

}
```

Trace
Debug
Info
Warn
Error
Fatal

Configuration

- log4j.xml(properties,yaml, json) – put to the classpath or set property –Dlog4j.configurationFile=<location>

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
    <Appenders>
        <ThresholdFilter level="DEBUG"/>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout
                pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />
        </Console>
    </Appenders>
    <Appenders>
    <RollingFile name="RollingFile" fileName="logs/app.log"
                 filePattern="logs/app-%d{MM-dd-yyyy}.log.gz">
      <PatternLayout>
        <pattern>%d %p %c{1.} [%t] %m%n</pattern>
      </PatternLayout>
      <TimeBasedTriggeringPolicy />
    </RollingFile>
  </Appenders>
    <Loggers>
        <Logger name="javaII.lab01.App" level="trace">
            <AppenderRef ref="Console" />
            <AppenderRef ref="RollingFile" />
        </Logger>
        <Root level="error">
            <AppenderRef ref="Console" />
        </Root>
    </Loggers>
</Configuration>
```

Marker

- Can use filter “MarkerFilter” for appender

```
Marker marker = MarkerManager.getMarker("tp-count");

log.info(marker, "sensitive info");
```

MDC (Mapped Diagnostic Context)

```
MDC.put("transaction.id", tx.getTransactionId());
MDC.put("transaction.owner", tx.getSender());
log4jBusinessService.transfer(tx.getAmount());
MDC.clear();
```

```
log4j.appender.consoleAppender.layout.ConversionPattern=
  %-4r [%t] %5p %c{1} %x - %m - tx.id=%X{transaction.id} tx.owner=%X{transaction.owner}%n
```

```
638  [pool-1-thread-2]  INFO Log4JBusinessService
  - Has transfer of 1104$ completed successfully ? true. - tx.id=2 tx.owner=Marc
638  [pool-1-thread-2]  INFO Log4JBusinessService
  - Preparing to transfer 1685$. - tx.id=4 tx.owner=John
```

Assertions

- They provide mechanism for *internal* consistency checks.
  - E.g. constraints among values of attributes is ensured.
- They could be removed in production version.
  - E.g. they are ignored during runtime.
- Java provides support with *assert* keyword.

Java Assertion Statement

- Two forms are used:
  - **`assert`** *boolean-expression*
  - **`assert`** *boolean-expression* **:** *description*
- The “boolean-expression” expresses something that should be true during its execution.
- An AssertionError is thrown if the assertion is false – it contains “description”

Assert example

```
public void removeRecord(String key)
{
    if(key == null){
        throw new IllegalArgumentException("...");
    }
    if(keyInUse(key)) {
        Record details = book.get(key);
        details.freeData();
        details.removeFromIndex();
        numberOfEntries--;
    }
    assert !keyInUse(key);
    assert isConsistentIndex() :
           "Inconsistent index in removeRecord";
}
```

Guidelines for Assertions

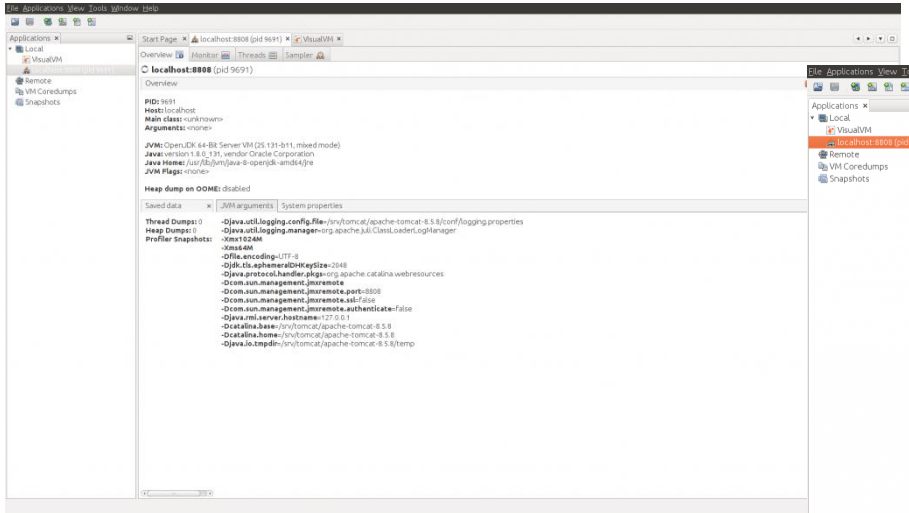
- Use it for internal consistency check.
- (Not?) Remove from production code.
- Don't include normal functionality:

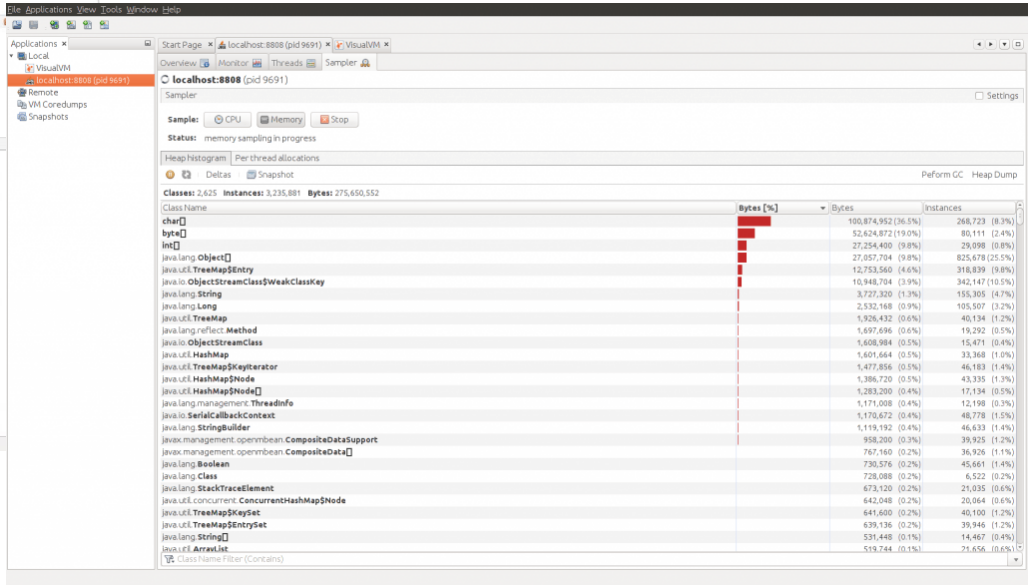
```
// Incorrect use:
assert book.remove(name) != null;
```

- They has to have no side effect.
- Do not use it for exception throwing – it is not an alternative

Profiling – Java VisualVM

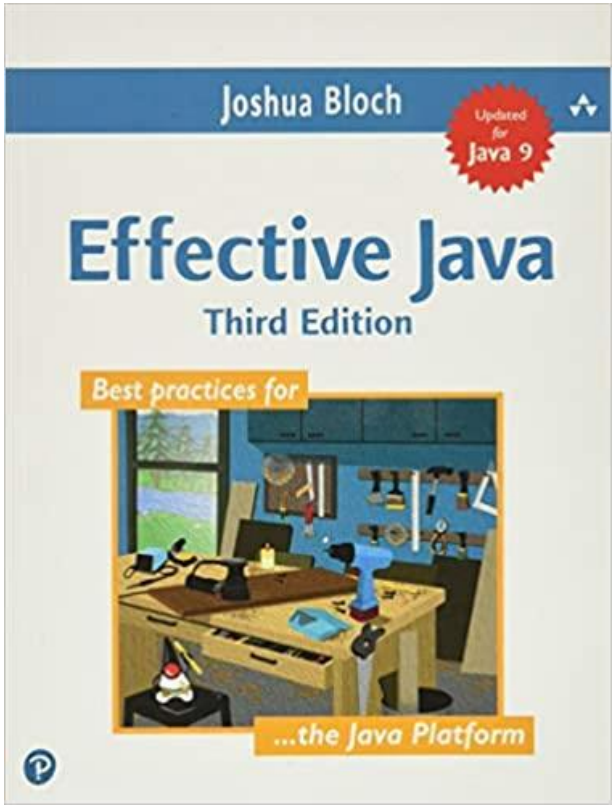
- Also part of JDK





Effective Java

- BLOCH, Joshua. Effective Java. 3rd edition. Boston: Addison-Wesley Professional, 2017. ISBN 978-0-13-468599-1.



Consider static factory methods instead of constructors

```
public static Boolean valueOf(boolean b) {
    return b ? Boolean.TRUE : Boolean.FALSE;
}
```

Consider static factory methods instead of constructors - advantages

- unlike constructors, they have names
- unlike constructors, they are not required to create a new object each time they're invoked
- unlike constructors, they can return an object of any subtype of their return type.
- class of the returned object can vary from call to call as a function of the input parameters
- the class of the returned object need not exist when the class containing the method is written.

Consider static factory methods instead of constructors - limitation

- classes without public or protected constructors cannot be subclassed;
- they are hard for programmers to find

Static factory methods name convention

- from
- of
- valueOf
- instance or getInstance
- create or newInstance
- getType
- newType
- type

```
Date d = Date.from(instant);

Set<Rank> faceCards
 = EnumSet.of(JACK, QUEEN, KING);

BigInteger prime
 = BigInteger.valueOf(Integer.MAX_VALUE);

StackWalker luke
 = StackWalker.getInstance(options);

Object newArray
 = Array.newInstance(classObject, arrayLen);

FileStore fs = Files.getFileStore(path);

BufferedReader br
 = Files.newBufferedReader(path);
```

Consider a builder when faced with many constructor parameters -telescoping constructors

- provide a constructor with only the required parameters

```
public NutritionFacts(int servingSize, int servings) {
        this(servingSize, servings, 0);
    }

    public NutritionFacts(int servingSize, int servings,
            int calories) {
        this(servingSize, servings, calories, 0);
    }
NutritionFacts cocaCola =
    new NutritionFacts(240, 8, 100, 0, 35, 27);

NutritionFacts cocaCola =
    new NutritionFacts(240, 8, 100, 0, 35, 27);
```

Consider a builder when faced with many constructor parameters – JavaBeans pattern

- JavaBean may be in an inconsistent state partway through its construction.
- the JavaBeans pattern precludes the possibility of making a class immutable

```
NutritionFacts cocaCola = new NutritionFacts();
cocaCola.setServingSize(240);
cocaCola.setServings(8);
cocaCola.setCalories(100);
cocaCola.setSodium(35);
cocaCola.setCarbohydrate(27);
```

Consider a builder when faced with many constructor parameters

- the client calls a constructor (or static factory) with all of the required parameters and gets a builder object.

```
NutritionFacts cocaCola = new NutritionFacts.Builder(240, 8)
        .calories(100).sodium(35).carbohydrate(27).build();
```

Obey the general contract when overriding equals

- No need to override:
  - Each instance of the class is inherently unique.
  - There is no need for the class to provide a “logical equality” test.
  - A superclass has already overridden equals, and the superclass behavior is appropriate for this class.
  - The class is private or package-private, and you are certain that its equals method will never be invoked.

Obey the general contract when overriding equals

- Signature … `public boolean equals(Object other)`
- Reflexive
- Symmetric
- Transitive
- Consistent
- For any non-null reference value x, x.equals(null) must return false

Obey the general contract when overriding equals – violated symmetry

```
public class Point {
     private final int x;
     private final int y;

    public Point(int x, int y) {
         this.x = x;
         this.y = y;
     }

    @Override public boolean equals(Object o) {
         if (!(o instanceof Point))
             return false;
         Point p = (Point)o;
         return p.x == x && p.y == y;
     }

    ...  // Remainder omitted
 }
```

```
public class ColorPoint extends Point {
     private final Color color;

    public ColorPoint(int x, int y, Color color) {
         super(x, y);
         this.color = color;
     }

    ...  // Remainder omitted
 }

// Broken - violates symmetry!
 @Override public boolean equals(Object o) {
     if (!(o instanceof ColorPoint))
        return false;
     return super.equals(o) && ((ColorPoint) o).color == color;
 }
Point p = new Point(1, 2);
 ColorPoint cp = new ColorPoint(1, 2, Color.RED);
```

Obey the general contract when overriding equals – violated transitivity

```
/ Broken - violates transitivity!
 @Override public boolean equals(Object o) {
    if (!(o instanceof Point))
        return false;

   // If o is a normal Point, do a color-blind comparison
    if (!(o instanceof ColorPoint))
        return o.equals(this);

   // o is a ColorPoint; do a full comparison
    return super.equals(o) && ((ColorPoint) o).color == color;
 }

ColorPoint p1 = new ColorPoint(1, 2, Color.RED);
 Point p2 = new Point(1, 2);
 ColorPoint p3 = new ColorPoint(1, 2, Color.BLUE);
```

Obey the general contract when overriding equals – violated Liskov subst. principle

```
// Broken - violates Liskov substitution principle (page 43)
 @Override public boolean equals(Object o) {
    if (o == null || o.getClass() != getClass())
       return false;
    Point p = (Point) o;
    return p.x == x && p.y == y;
 }

public class ColorPoint {
   private final Point point;
   private final Color color;

  public ColorPoint(int x, int y, Color color) {
      point = new Point(x, y);
      this.color = Objects.requireNonNull(color);
   }
```

```
/**
    * Returns the point-view of this color point.
    */
   public Point asPoint() {
      return point;
   }

  @Override public boolean equals(Object o) {
      if (!(o instanceof ColorPoint))
         return false;
      ColorPoint cp = (ColorPoint) o;
      return cp.point.equals(point) && cp.color.equals(color);
   }

  ...   // Remainder omitted
 }
```

Obey the general contract when overriding equals

```
public Point{

        @Override
         public boolean equals(Object o) {
             if (o instanceof Point p)
                return p.canEqual(this) && …;
            }
            return false;
         }



  ...   // Remainder omitted
         public boolean canEqual(Point that) {
                  return that instanceOf Point;
         }
}
```

```
public class ColorPoint extends Point{
   private final Point point;
   private final Color color;

  public ColorPoint(int x, int y, Color color) {
     super(x, y);
     this.color = Objects.requireNonNull(color);
   }

  @Override
   public boolean equals(Object o) {
     if (!(o instanceof ColorPoint))
       return false;
     ColorPoint cp = (ColorPoint) o;
     return super.equals(cp) && cp.color.equals(color);
  }

  ...     // Remainder omitted
   @Overrid
    public boolean canEqual(Point that)
       return that instanceOf ColorPoint;
    }
}
```

Always override hashCode when you override equals

1. Declare an int variable named result, and initialize it to the hash code c for the first significant field in your object, as computed in step 2.1
2. For every remaining significant field f in your object, do the following:
   a) Compute an int hash code c for the field:
      1. If the field is of a primitive type, compute *Type*.hashCode(f), where *Type* is the boxed primitive class corresponding to f's type.
      2. If the field is an object reference and this class's equals method compares the field by recursively invoking equals, recursively invoke hashCode on the field. If a more complex comparison is required, compute a "canonical representation" for this field and invoke hashCode on the canonical representation. If the value of the field is null, use 0 (or some other constant, but 0 is traditional).
      3. If the field is an array, treat it as if each significant element were a separate field. That is, compute a hash code for each significant element by applying these rules recursively, and combine the values per step 2.b. If the array has no significant elements, use a constant, preferably not 0. If all elements are significant, use Arrays.hashCode.
   b) Combine the hash code c computed in step 2.a into result as follows:

      result = 31 * result + c
3. Return result.

Always override hashCode when you override equals

- Do not be tempted to exclude significant fields from the hash code computation to improve performance.

3rd lecture

- Effective java II

Minimize the accessibility of classes and members

- make each class or member as inaccessible as possible
- Instance fields of public classes should rarely be public
- classes with public mutable fields are not generally thread-safe
- it is wrong for a class to have a public static final array field, or an accessor that returns such a field

Minimize mutability

- Don’t provide methods that modify the object’s state (known as mutators).
- Ensure that the class can’t be extended
- Make all fields final.
- Make all fields private.
- Ensure exclusive access to any mutable components.

Favor composition over inheritance

```
// Broken - Inappropriate use of inheritance!
 public class InstrumentedHashSet<E> extends HashSet<E> {
     // The number of attempted element insertions
     private int addCount = 0;

    public InstrumentedHashSet() {
     }

    public InstrumentedHashSet(int initCap, float loadFactor) {
         super(initCap, loadFactor);
     }
     @Override public boolean add(E e) {
         addCount++;
         return super.add(e);
     }
     @Override public boolean addAll(Collection<? extends E> c) {
         addCount += c.size();
         return super.addAll(c);
     }
     public int getAddCount() {
         return addCount;
     }
 }
```

Favor composition over inheritance

```
// Wrapper class - uses composition in place of inheritance
 public class InstrumentedSet<E> extends ForwardingSet<E> {
     private int addCount = 0;

    public InstrumentedSet(Set<E> s) {
         super(s);
     }

    @Override public boolean add(E e) {
         addCount++;
         return super.add(e);
      }

     @Override public boolean addAll(Collection<? extends E> c) {
          addCount += c.size();
          return super.addAll(c);
      }
      public int getAddCount() {
          return addCount;
      }
 }
```

Favor composition over inheritance

```
// Reusable forwarding class
 public class ForwardingSet<E> implements Set<E> {
     private final Set<E> s;
     public ForwardingSet(Set<E> s) { this.s = s; }

    public void clear()               { s.clear();            }
     public boolean contains(Object o) { return s.contains(o); }
     public boolean isEmpty()          { return s.isEmpty();   }
     public int size()                 { return s.size();      }
     public Iterator<E> iterator()     { return s.iterator();  }
     public boolean add(E e)           { return s.add(e);      }
     public boolean remove(Object o)   { return s.remove(o);   }
     public boolean containsAll(Collection<?> c)
                                   { return s.containsAll(c); }
     public boolean addAll(Collection<? extends E> c)
                                   { return s.addAll(c);      }
     public boolean removeAll(Collection<?> c)
                                   { return s.removeAll(c);   }
     public boolean retainAll(Collection<?> c)
                                   { return s.retainAll(c);   }
     public Object[] toArray()          { return s.toArray();  }
     public <T> T[] toArray(T[] a)      { return s.toArray(a); }
     @Override public boolean equals(Object o)
                                        { return s.equals(o);  }
     @Override public int hashCode()    { return s.hashCode(); }
     @Override public String toString() { return s.toString(); }
 }
```

Use enums instead of int constants

- The int enum pattern - severely deficient!

```
public enum Apple  { FUJI, PIPPIN, GRANNY_SMITH }
 public enum Orange { NAVEL, TEMPLE, BLOOD }
```

```
public static final int APPLE_FUJI        = 0;
public static final int APPLE_PIPPIN      = 1;
public static final int APPLE_GRANNY_SMITH = 2;
public static final int ORANGE_NAVEL  = 0;
public static final int ORANGE_TEMPLE = 1;
public static final int ORANGE_BLOOD  = 2;
```

Enum type with data and behavior

- To associate data with enum constants, declare instance fields and write a constructor that takes the data and stores it in the fields.

```
public enum Planet {
    MERCURY(3.302e+23, 2.439e6),
    VENUS  (4.869e+24, 6.052e6),
    EARTH  (5.975e+24, 6.378e6),
    MARS   (6.419e+23, 3.393e6),
    JUPITER(1.899e+27, 7.149e7),
    SATURN (5.685e+26, 6.027e7),
    URANUS (8.683e+25, 2.556e7),
    NEPTUNE(1.024e+26, 2.477e7);

   private final double mass;           // In kilograms
    private final double radius;         // In meters
    private final double surfaceGravity; // In m / s^2

   // Universal gravitational constant in m^3 / kg s^2
    private static final double G = 6.67300E-11;
```

```
    // Constructor
     Planet(double mass, double radius) {
         this.mass = mass;
         this.radius = radius;
         surfaceGravity = G * mass / (radius * radius);
     }

    public double mass()           { return mass; }
     public double radius()         { return radius; }
     public double surfaceGravity() { return surfaceGravity; }

    public double surfaceWeight(double mass) {
         return mass * surfaceGravity;  // F = ma
     }
}
```

Enum with different behavior

- Enum type that switches on its own value - questionable

```
public enum Operation {
    PLUS, MINUS, TIMES, DIVIDE;

   // Do the arithmetic operation represented by this constant
    public double apply(double x, double y) {
        switch(this) {
            case PLUS:   return x + y;
            case MINUS:  return x - y;
            case TIMES:  return x * y;
            case DIVIDE: return x / y;
        }
        throw new AssertionError("Unknown op: " + this);
    }
}
```

Enum with different behavior II

- Enum type with constant-specific class bodies and data

```
public enum Operation {
     PLUS("+") {
         public double apply(double x, double y) { return x + y; }
     },
     MINUS("-") {
         public double apply(double x, double y) { return x - y; }
     },
     TIMES("*") {
         public double apply(double x, double y) { return x * y; }
     },
     DIVIDE("/") {
         public double apply(double x, double y) { return x / y; }
     };

    private final String symbol;

    Operation(String symbol) { this.symbol = symbol; }

    @Override public String toString() { return symbol; }

    public abstract double apply(double x, double y);
}
```

Implementing a fromString method on an enum type

```
 private static final Map<String, Operation> stringToEnum =
        Stream.of(values()).collect(
            toMap(Object::toString, e -> e));

// Returns Operation for string, if any
 public static Optional<Operation> fromString(String symbol) {
     return Optional.ofNullable(stringToEnum.get(symbol));
 }
```

Enum that switches on its value to share code – questionable

```
enum PayrollDay {
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY,
    SATURDAY, SUNDAY;

   private static final int MINS_PER_SHIFT = 8 * 60;

   int pay(int minutesWorked, int payRate) {
        int basePay = minutesWorked * payRate;

       int overtimePay;
        switch(this) {
          case SATURDAY: case SUNDAY: // Weekend
            overtimePay = basePay / 2;
            break;
          default: // Weekday
            overtimePay = minutesWorked <= MINS_PER_SHIFT ?
              0 : (minutesWorked - MINS_PER_SHIFT) * payRate / 2;
        }

       return basePay + overtimePay;
    }
}
```

The strategy enum pattern

```
// The strategy enum pattern
 enum PayrollDay {
     MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY,
     SATURDAY(PayType.WEEKEND), SUNDAY(PayType.WEEKEND);

    private final PayType payType;

    PayrollDay(PayType payType) { this.payType = payType; }
     PayrollDay() { this(PayType.WEEKDAY); }  // Default

    int pay(int minutesWorked, int payRate) {
         return payType.pay(minutesWorked, payRate);
     }
```

```
// The strategy enum type
    private enum PayType {
        WEEKDAY {
            int overtimePay(int minsWorked, int payRate) {
                return minsWorked <= MINS_PER_SHIFT ? 0 :
                  (minsWorked - MINS_PER_SHIFT) * payRate / 2;
            }
        },
        WEEKEND {
            int overtimePay(int minsWorked, int payRate) {
                return minsWorked * payRate / 2;
            }
        };

        abstract int overtimePay(int mins, int payRate);
         private static final int MINS_PER_SHIFT = 8 * 60;

        int pay(int minsWorked, int payRate) {
            int basePay = minsWorked * payRate;
            return basePay + overtimePay(minsWorked, payRate);
         }
    }
}
```

Enum with switches

Switches on enums are good for augmenting enum types with constant-specific behavior.

```
public static Operation inverse(Operation op) {
    switch(op) {
        case PLUS:   return Operation.MINUS;
        case MINUS:  return Operation.PLUS;
        case TIMES:  return Operation.DIVIDE;
        case DIVIDE: return Operation.TIMES;

        default:  throw new AssertionError("Unknown op: " + op);
    }
}
```

Using enums

- Use enums any time you need a set of constants whose members are known at compile time.
- It is not necessary that the set of constants in an enum type stay fixed for all time.

Prefer lambdas to anonymous classes

```
// Anonymous class instance as a function object - obsolete!
 Collections.sort(words, new Comparator<String>() {
     public int compare(String s1, String s2) {
         return Integer.compare(s1.length(), s2.length());
     }
 });

// Lambda expression as function object (replaces anonymous class)
 Collections.sort(words,
         (s1, s2) -> Integer.compare(s1.length(), s2.length()));
```

Lambdas

- Omit the types of all lambda parameters unless their presence makes your program clearer.
- if a computation isn't self-explanatory, or exceeds a few lines, don't put it in a lambda
- Don't use anonymous classes for function objects unless you have to create instances of types that aren't functional interfaces.

Prefer method references to lambdas

Method Ref Type	Example	Lambda Equivalent
Static	Integer::parseInt	str -> Integer.parseInt(str)
Bound	Instant.now()::isAfter	Instant then = Instant.now(); t -> then.isAfter(t)
Unbound	String::toLowerCase	str -> str.toLowerCase()
Class Constructor	TreeMap<K,V>::new	() -> new TreeMap<K,V>
Array Constructor	int[]::new	len -> new int[len]

Prefer method references to lambdas

- Where method references are shorter and clearer, use them; where they aren't, stick with lambdas.

```
//lambda
map.merge(key, 1, (count, incr) -> count + incr);


//method reference
map.merge(key, 1, Integer::sum);
```

Favor the use of standard functional interfaces

Interface	Function Signature	Example
UnaryOperator<T>	`T apply(T t)`	`String::toLowerCase`
BinaryOperator<T>	`T apply(T t1, T t2)`	`BigInteger::add`
Predicate<T>	`boolean test(T t)`	`Collection::isEmpty`
Function<T,R>	`R apply(T t)`	`Arrays::asList`
Supplier<T>	`T get()`	`Instant::now`
Consumer<T>	`void accept(T t)`	`System.out::println`

Functional interface

- If one of the standard functional interfaces does the job, you should generally use it in preference to a purpose-built functional interface.
- Don't be tempted to use basic functional interfaces with boxed primitives instead of primitive functional interfaces.
- Always annotate your functional interfaces with the `@FunctionalInterface` annotation.

Use streams judiciously

Prefer side-effect-free functions in streams

Prefer Collection to Stream as a return type

Use caution when making streams parallel

Check parameters for validity

```
/**
 * Returns a BigInteger whose value is (this mod m). This method
 * differs from the remainder method in that it always returns a
 * non-negative BigInteger.
 *
 * @param m the modulus, which must be positive
 * @return this mod m
 * @throws ArithmeticException if m is less than or equal to 0
 */
public BigInteger mod(BigInteger m) {
   if (m.signum() <= 0)
      throw new ArithmeticException("Modulus <= 0: " + m);
   ... // Do the computation
}
```

Check null, ranges

- Another methods `checkFromIndexSize`, `checkFromToIndex`, `chekcIndex`

```
// Inline use of Java's null-checking facility
this.strategy = Objects.requireNonNull(strategy, "strategy");
```

Make defensive copies when needed

- You must program defensively, with the assumption that clients of your class will do their best to destroy its invariants.

Mutable parameters in constructor

- it is essential to make a defensive copy of each mutable parameter to the constructor
- defensive copies are made before checking the validity of the parameters and the validity check is performed on the copies rather than on the originals

```
// Repaired constructor - makes defensive copies of parameters
public Period(Date start, Date end) {
    this.start = new Date(start.getTime());
    this.end   = new Date(end.getTime());

    if (this.start.compareTo(this.end) > 0)
      throw new IllegalArgumentException(
          this.start + " after " + this.end);
}
```

Mutable return values

- return defensive copies of mutable internal fields

```
public Date start() {
    return new Date(start.getTime());
}

public Date end() {
    return new Date(end.getTime());
}
```

Mutable return values

- Wrap mutable return values with unmutable wrappers

```
public Collection<Objects> getCollections() {
  return Collections.unmodifiableCollection(collections);
}
```

Return empty collections or arrays, not nulls

- never return null in place of an empty array or collection

```
List<Cheese> cheeses = shop.getCheeses();
if (cheeses != null && cheeses.contains(Cheese.STILTON))
    System.out.println("Jolly good, just the thing.");
```

```
return Collections.emptyList();
```

4th lecture

- Internationalization
- Big values
- Money in Java

Internationalization

- Support for different character sets and for universal (UTF-8, UTF-16)
- Support for specific settings: format (number, date, ….), currencies, texts and other resources (multimedia, data)
- *Locale* is a class that identifies a combination of language and region:
  - *Locale(String language)*
  - *Locale(String language, String country)*

```
Locale czechLocale = new Locale(“cs”,”CZ”)
```

String localization

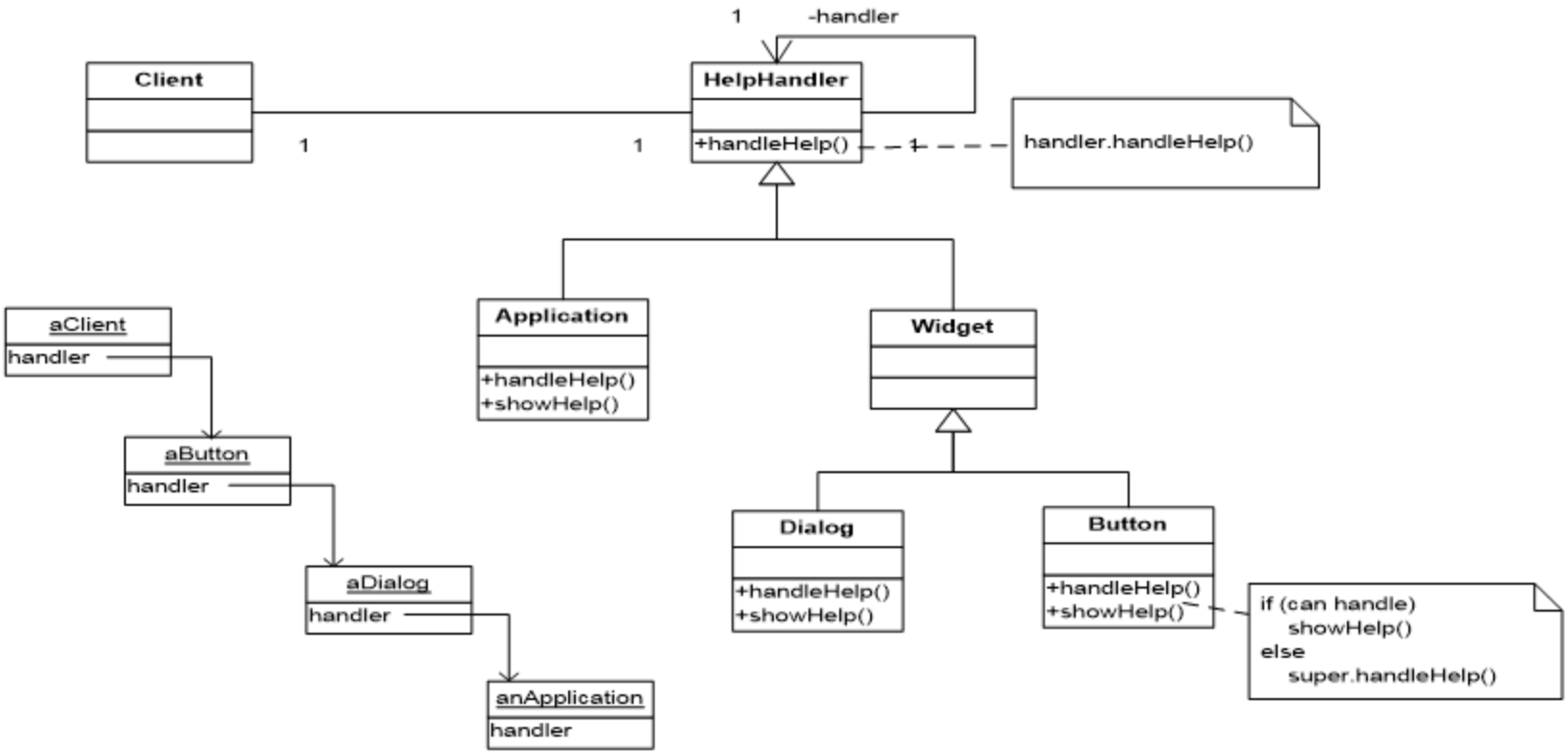
```
ResourceBundle bundle = ResourceBundle.getBundle("MessageBundle", locale);
System.out.println("" + bundle.getString("greetings"));
```

- Files in resources directory:
  - MessageBundle.properties
    `greetings = Hello`
  - MessageBundle_de.properties
    `greetings = Hallo`
  - MessageBundle_fr.properties
    `greetings = Bonjour`
  - MessageBundle_it.properties
    `greetings =` Ciao
- For German, French and Italian is used given text and default for other (Hello).

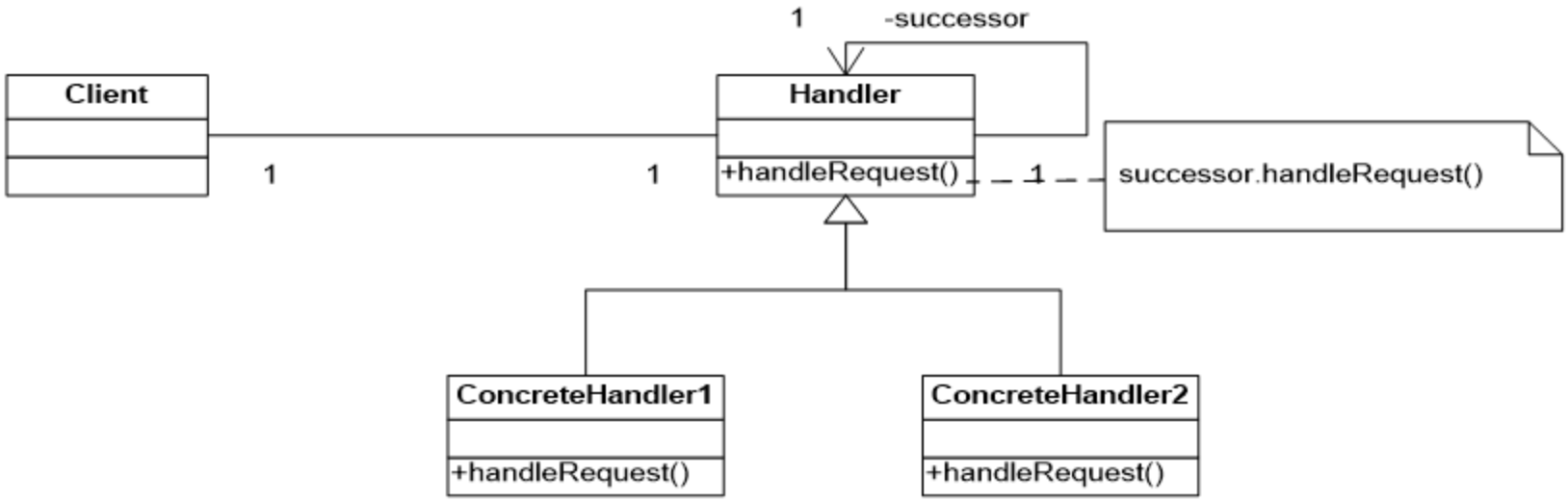
Chain of Responsibility Design Pattern

- It avoids coupling senders of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

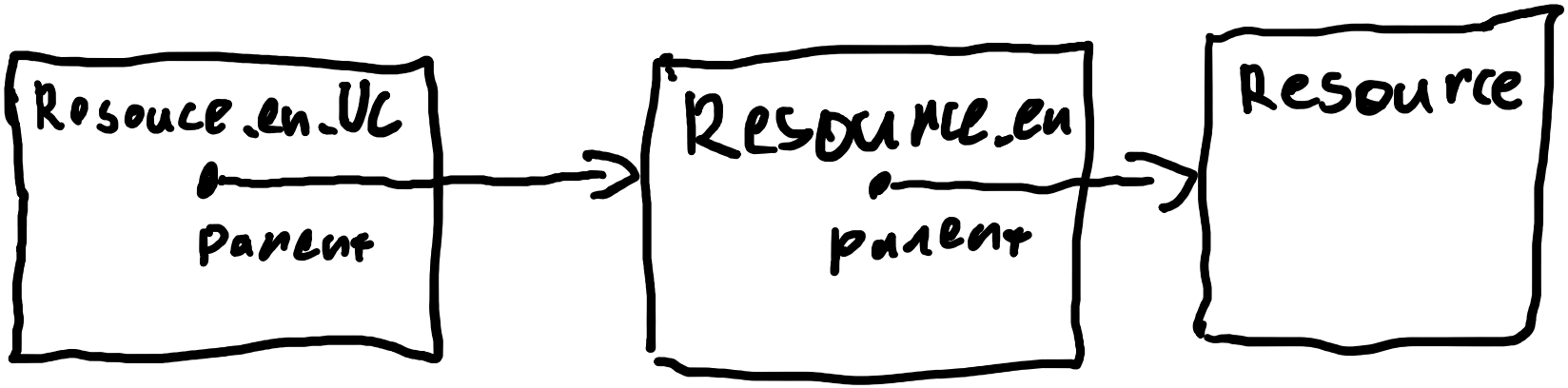
Chain of Responsibility Example



Chain of Responsibility Design Pattern



Chain of Responsibility – resource bundles



Formatting (with predefined format)

- Numbers:

```
NumberFormat nf = NumberFormat.getNumberInstance(new Locale("fr"));
String valueStr = nf.format(Math.PI);
System.out.println(valueStr);
-----------------------
Output is:3,142
```

- Currency:

```
Locale locale_enGB = new Locale("en","GB");
Currency currency = Currency.getInstance(locale_enGB);
NumberFormat currencyFormat = NumberFormat.getCurrencyInstance(locale_enGB);
System.out.println(currency.getDisplayName() + ": " + currencyFormat.format(100.0));
-----------------------------------------
Output is: British Pound: £100.00
```

- Datetime

```
DateTimeFormatter dateTimeFormat = DateTimeFormatter.ofLocalizedDateTime(FormatStyle.FULL).withLocale(new Locale("cs", "CZ"));
System.out.println(dateTimeFormat.format(ZonedDateTime.now()));
-----------------------------------------------------------------------------------------------------
Output is:čtvrtek 21. listopadu 2019 15:33:34 Středoevropský standardní čas
```

Big values

- **Double** does not have unlimited precision

```
double val = 0.1;
for (int i = 0; i < 10; i++) {
          val += 0.1;
}
System.out.printf("val = " + val);

--- output

val = 1.0999999999999999
```

BigDecimal and BigInteger

- https://www.baeldung.com/java-bigdecimal-biginteger

BigDecimal

- BigDecimal represents an immutable arbitrary-precision signed decimal number.
  - Unscaled value
  - Scale (32 bit)
- High-precision arithmetic
- Variety constructors(String, character array, int, long, and BigInteger) and factory method *valueOf* (*double, long*)

BigDecimal operations

- Arithmetic operation - `add, subtract, multiply, divide, …`
- Relational operation – compareTo, equals (compares also scale)
- Functions – abs, pow, sqrt,..
- Various attributes – precision, scale, sign
- Rounding – 8 modes

BigInteger

- immutable arbitrary-precision integers
- used when integers involved are larger than the limit of long type
- Constructor (String, byte array) and *valueOf* (long)

BigInteger operations

- Similar to *int* and *long* but cannot overflow
  - *Arithmetic, bitwise,* - as methods
- Bit manipulation methods
- GCD, modular arithmetic, prime generation, primality testing,

Money in Java

“A large proportion of the computers in this world manipulate money, so it's always puzzled me that money isn't actually a first class data type in any mainstream programming language. “

Martin Fowler

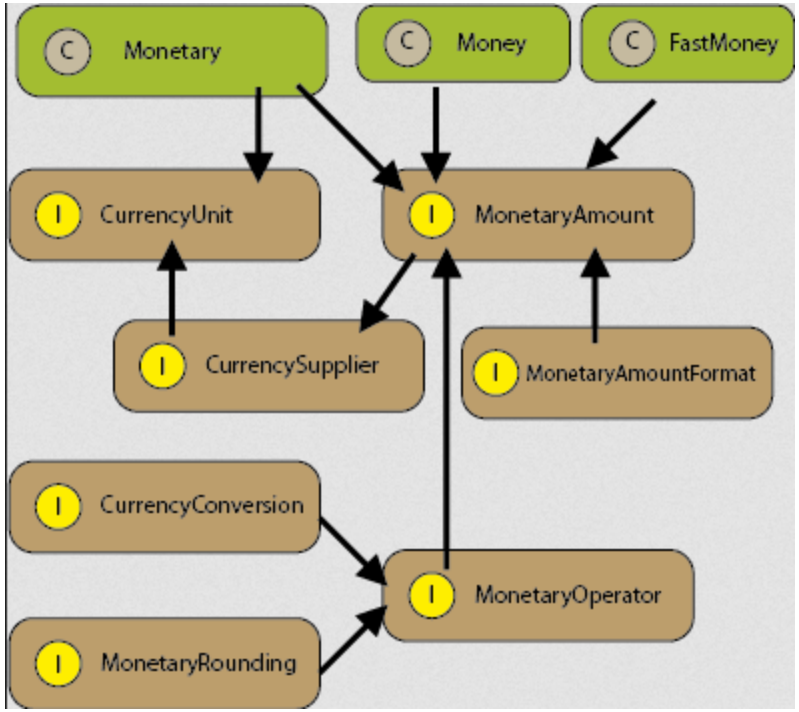
Money in Java - standards

- Joda money
- JSR 354

JSR 354

- To provide an API for handling and calculating monetary amounts
- To define classes representing currencies and monetary amounts, as well as monetary rounding
- To deal with currency exchange rates
- To deal with formatting and parsing of currencies and monetary amounts

JSR 354 - model



JSR 354 - *CurrencyUnit*

```
@Test
public void givenCurrencyCode_whenString_thanExist() {
    CurrencyUnit usd = Monetary.getCurrency("USD");

    assertNotNull(usd);
    assertEquals(usd.getCurrencyCode(), "USD");
    assertEquals(usd.getNumericCode(), 840);
    assertEquals(usd.getDefaultFractionDigits(), 2);
}
```

```
@Test(expected = UnknownCurrencyException.class)
public void givenCurrencyCode_whenNoExist_thanThrowsError() {
    Monetary.getCurrency("AAA");
}
```

JSR 354 - *MonetaryAmount*

```
@Test
public void givenAmounts_whenStringified_thanEquals() {

    CurrencyUnit usd = Monetary.getCurrency("USD");
    MonetaryAmount fstAmtUSD = Monetary.getDefaultAmountFactory()
      .setCurrency(usd).setNumber(200).create();
    Money moneyof = Money.of(12, usd);
    FastMoney fastmoneyof = FastMoney.of(2, usd);

    assertEquals("USD", usd.toString());
    assertEquals("USD 200", fstAmtUSD.toString());
    assertEquals("USD 12", moneyof.toString());
    assertEquals("USD 2.00000", fastmoneyof.toString());
}
```

JSR 354 – Monetary Arithmetic

```
@Test
public void givenCurrencies_whenCompared_thanNotequal() {
    MonetaryAmount oneDolar = Monetary.getDefaultAmountFactory()
      .setCurrency("USD").setNumber(1).create();
    Money oneEuro = Money.of(1, "EUR");

    assertFalse(oneEuro.equals(FastMoney.of(1, "EUR")));
    assertTrue(oneDolar.equals(Money.of(1, "USD")));
}
```

```
@Test
public void givenAmounts_whenSummed_thanCorrect() {
    MonetaryAmount[] monetaryAmounts = new MonetaryAmount[] {
      Money.of(100, "CHF"), Money.of(10.20, "CHF"), Money.of(1.15, "CHF")};

    Money sumAmtCHF = Money.of(0, "CHF");
    for (MonetaryAmount monetaryAmount : monetaryAmounts) {
        sumAmtCHF = sumAmtCHF.add(monetaryAmount);
    }

    assertEquals("CHF 111.35", sumAmtCHF.toString());
}
```

JSR 354 – Monetary Rounding

```
@Test
public void givenAmount_whenRounded_thanEquals() {
    MonetaryAmount fstAmtEUR = Monetary.getDefaultAmountFactory()
      .setCurrency("EUR").setNumber(1.30473908).create();
    MonetaryAmount roundEUR = fstAmtEUR.with(Monetary.getDefaultRounding());

    assertEquals("EUR 1.30473908", fstAmtEUR.toString());
    assertEquals("EUR 1.3", roundEUR.toString());
}
```

JSR 354 – Currency Conversion

```
@Test
public void givenAmount_whenConversion_thenNotNull() {
    MonetaryAmount oneDollar = Monetary.getDefaultAmountFactory().setCurrency("USD")
      .setNumber(1).create();

    CurrencyConversion conversionEUR = MonetaryConversions.getConversion("EUR");

    MonetaryAmount convertedAmountUSDtoEUR = oneDollar.with(conversionEUR);

    assertEquals("USD 1", oneDollar.toString());
    assertNotNull(convertedAmountUSDtoEUR);
}
```

JSR 354 - Formatting

```
@Test
public void givenLocale_whenFormatted_thanEquals() {
    MonetaryAmount oneDollar = Monetary.getDefaultAmountFactory()
      .setCurrency("USD").setNumber(1).create();

    MonetaryAmountFormat formatUSD = MonetaryFormats.getAmountFormat(Locale.US);
    String usFormatted = formatUSD.format(oneDollar);

    assertEquals("USD 1", oneDollar.toString());
    assertNotNull(formatUSD);
    assertEquals("USD1.00", usFormatted);
}
```

JSR 354 – Formatting II

```
@Test
public void givenAmount_whenCustomFormat_thanEquals() {
    MonetaryAmount oneDollar = Monetary.getDefaultAmountFactory()
            .setCurrency("USD").setNumber(1).create();

    MonetaryAmountFormat customFormat = MonetaryFormats.getAmountFormat(AmountFormatQueryBuilder.
      of(Locale.US).set(CurrencyStyle.NAME).set("pattern", "00000.00 ¤").build());
    String customFormatted = customFormat.format(oneDollar);

    assertNotNull(customFormat);
    assertEquals("USD 1", oneDollar.toString());
    assertEquals("00001.00 US Dollar", customFormatted);
}
```

5th Lecture

- Lock objects
- Executors
- Concurrent collections
- Atomic variables
- ThreadLocalRandom
- CompletableFuture
- Java NIO (New IO) I

Concurrency - references

- https://docs.oracle.com/javase/tutorial/essential/concurrency/forkjoin.html

Lock object

- Similar to mechanism of synchronized sections
- Can back out of and attempt to acquire lock - *tryLock*

Executors

- Executor interfaces
- Thread Pool
- Fork/Join

Executors – Executor Interfaces

- `Executor` - a simple interface that supports launching new tasks.
- `ExecutorService` - a subinterface of Executor, which adds features that help manage the lifecycle, both of the individual tasks and of the executor itself.

- `ScheduledExecutorService` - a subinterface of ExecutorService, supports future and/or periodic execution of tasks.

Executor

- Behavior depends on implementation.

```
(new Thread(r)).start();

//------------------------

e.execute(r);
```

ExecutorService

- extends *Executor:*
  - shutdown()
  - shutdownNow()
  - isShutdown()
  - isTerminated()
  - awaitTermination(long, TimeUnit)
  - submit(Callable<T>)
  - submit(Runnable, T)
  - submit(Runnable)
  - invokeAll(Collection<? extends Callable<T>>)
  - invokeAll(Collection<? extends Callable<T>>, long, TimeUnit)
  - invokeAny(Collection<? extends Callable<T>>)
  - invokeAny(Collection<? extends Callable<T>>, long, TimeUnit)

ScheduledExecutorService

- extension of ExecutorService
  - schedule
  - scheduleAtFixedRate
  - scheduleWithFixedDelay

Thread Pools

- Consist of worker threads – can be used to execute multiple tasks
- Minimizes the overhead due thread creation
- Factory methods in java.util.concurrent.Executors:
  - newFixedThreadPool
  - newCachedThreadPool
  - newSingleThreadExecutor
  - Versions with ScheduledExecutorService
- `java.util.concurrent.ThreadPoolExecutor` and `java.util.concurrent.ScheduledThreadPoolExecutor`

Fork/Join

- Implementation of `ExecutorService` – designed for work that could be broken into smaller pieces
- The goal is to use all the available processing power
- Distinct from a thread pool implementing *work-stealing* algorithm
- The main class is `ForkJoinPool`

Fork/join basic scheme usage

```
if (my portion of the work is small enough)
  do the work directly
else
  split my work into two pieces
  invoke the two pieces and wait for the results
```

Fork/join – example

```
public class ForkBlur extends RecursiveAction {
    private int[] mSource;
    private int mStart;
    private int mLength;
    private int[] mDestination;

    // Processing window size; should be odd.
    private int mBlurWidth = 15;

    public ForkBlur(int[] src, int start, int length, int[] dst) {
        mSource = src;
        mStart = start;
        mLength = length;
        mDestination = dst;
    }

    protected void computeDirectly() {
// …
    }
```

Fork/join – compute method

```
protected static int sThreshold = 100000;

protected void compute() {
    if (mLength < sThreshold) {
        computeDirectly();
        return;
    }

    int split = mLength / 2;

    invokeAll(new ForkBlur(mSource, mStart, split, mDestination),
              new ForkBlur(mSource, mStart + split, mLength - split,
                           mDestination));
}
```

Fork/join – run example

1. Create a task that represents all of the work to be done.

```
// source image pixels are in src
// destination image pixels are in dst
ForkBlur fb = new ForkBlur(src, 0, src.length, dst);
```

2. Create the ForkJoinPool that will run the task.

```
ForkJoinPool pool = new ForkJoinPool();
```

3. Run the task.

```
pool.invoke(fb);
```

Concurrent Collections

- `BlockingQueue` – blocks or times out when full/empty during addition/retrieving
- `ConcurentMap` – interface that defines useful atomic operation
- `ConcurentNavigableMap` – extends *ConcurrentMap*

Atomic Variables

- Provides thread-safe operation for variable holding and modification
- AtomicInteger, AtomicLong, …

ThreadLocalRandom

- For concurrent access its using provides better performance

```
int r = ThreadLocalRandom.current() .nextInt(4, 77);
```

CompletableFuture

- https://www.callicoder.com/java-8-completablefuture-tutorial/

CompletableFuture - description

- extends `Future` – methods `done`, `isDone` – with:
  - can be manually completed,
  - man perform further action on a CompletableFuture's result without blocking,
  - multiple CompletableFutures can be chained together,
  - multiple CompletableFutures can be combined together,
  - exception handling.

Creating CompletableFuture with constructor

```
CompletableFuture<String> completableFuture = new CompletableFuture<String>();

String result = completableFuture.get()

completableFuture.complete("Future's Result")
```

Factory methods in CompletableFuture - runAsync

```
// Run a task specified by a Runnable Object asynchronously.
CompletableFuture<Void> future = CompletableFuture.runAsync(new Runnable() {
    @Override
    public void run() {
        // Simulate a long-running Job
        try {
            TimeUnit.SECONDS.sleep(1);
        } catch (InterruptedException e) {
            throw new IllegalStateException(e);
        }
        System.out.println("I'll run in a separate thread than the main thread.");
    }
});

// Block and wait for the future to complete
future.get()
```

Factory methods in CompletableFuture - supplyAsync

```
// Run a task specified by a Supplier object asynchronously
CompletableFuture<String> future = CompletableFuture.supplyAsync(new Supplier<String>() {
    @Override
    public String get() {
        try {
            TimeUnit.SECONDS.sleep(1);
        } catch (InterruptedException e) {
            throw new IllegalStateException(e);
        }
        return "Result of the asynchronous computation";
    }
});

// Block and get the result of the Future
String result = future.get();
System.out.println(result);
```

CompletableFuture – use Executor

```
// Variations of runAsync() and supplyAsync() methods
```

- `static CompletableFuture<Void>  runAsync(Runnable runnable)`
- `static CompletableFuture<Void>  runAsync(Runnable runnable, Executor executor)`
- `static <U> CompletableFuture<U> supplyAsync(Supplier<U> supplier)`
- `static <U> CompletableFuture<U> supplyAsync(Supplier<U> supplier, Executor executor)`

Transforming and acting on a CompletableFuture

- thenApply, thenAccept, thenRun

```
// Create a CompletableFuture
CompletableFuture<String> whatsYourNameFuture =
CompletableFuture.supplyAsync(() -> {
   try {
       TimeUnit.SECONDS.sleep(1);
   } catch (InterruptedException e) {
       throw new IllegalStateException(e);
   }
   return "Rajeev";
});

// Attach a callback to the Future using thenApply()
CompletableFuture<String> greetingFuture =
whatsYourNameFuture.thenApply(name -> {
   return "Hello " + name;
});

// Block and get the result of the future.
System.out.println(greetingFuture.get()); // Hello Rajeev
```

Combine two dependent futures using thenCompose()

```
CompletableFuture<User> getUsersDetail(String userId) {
          return CompletableFuture.supplyAsync(() -> {
                    return UserService.getUserDetails(userId);
          });
}

CompletableFuture<Double> getCreditRating(User user) {
          return CompletableFuture.supplyAsync(() -> {
                    return CreditRatingService.getCreditRating(user);
          });
}

CompletableFuture<Double> result = getUserDetail(userId)
.thenCompose(user -> getCreditRating(user));
```

Combine two independent futures using thenCombine()

```
System.out.println("Retrieving weight.");
CompletableFuture<Double> weightInKgFuture = CompletableFuture.supplyAsync(() -> {
    try {
        TimeUnit.SECONDS.sleep(1);
    } catch (InterruptedException e) {
       throw new IllegalStateException(e);
    }
    return 65.0;
});

System.out.println("Retrieving height.");
CompletableFuture<Double> heightInCmFuture = CompletableFuture.supplyAsync(() -> {
    try {
        TimeUnit.SECONDS.sleep(1);
    } catch (InterruptedException e) {
       throw new IllegalStateException(e);
    }
    return 177.8;
});

System.out.println("Calculating BMI.");
CompletableFuture<Double> combinedFuture = weightInKgFuture
        .thenCombine(heightInCmFuture, (weightInKg, heightInCm) -> {
    Double heightInMeter = heightInCm/100;
    return weightInKg/(heightInMeter*heightInMeter);
});

System.out.println("Your BMI is - " + combinedFuture.get())
```

Combining multiple CompletableFutures together

```
static CompletableFuture<Void>    allOf(CompletableFuture<?>... cfs)

static CompletableFuture<Object> anyOf(CompletableFuture<?>... cfs)
```

Handle exceptions using exceptionally() callback

```
Integer age = -1;

CompletableFuture<String> maturityFuture = CompletableFuture.supplyAsync(() ->
{
    if(age < 0) {
        throw new IllegalArgumentException("Age can not be negative");
    }
    if(age > 18) {
        return "Adult";
    } else {
        return "Child";
    }
}).exceptionally(ex -> {
    System.out.println("Oops! We have an exception - " + ex.getMessage());
    return "Unknown!";
});

System.out.println("Maturity : " + maturityFuture.get());
```

Handle exceptions using the generic handle() method

```
Integer age = -1;

CompletableFuture<String> maturityFuture = CompletableFuture.supplyAsync(() -> {
    if(age < 0) {
        throw new IllegalArgumentException("Age can not be negative");
    }
    if(age > 18) {
        return "Adult";
    } else {
        return "Child";
    }
}).handle((res, ex) -> {
    if(ex != null) {
        System.out.println("Oops! We have an exception - " + ex.getMessage());
        return "Unknown!";
    }
    return res;
});

System.out.println("Maturity : " + maturityFuture.get());
```

References

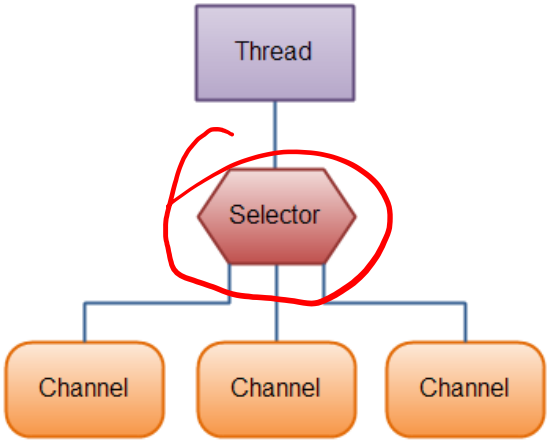
http://tutorials.jenkov.com/java-nio/index.html

Java NIO: Basic architecture

- Overview
- Channel
- Buffer
- Scatter/Gather
- Channel to Channel transfer
- Selector

Java NIO: Overview

- It enables non-blocking IO
- Core components:
  - Channels
  - Buffers
  - Selectors



Java NIO: Channels

- FileChannel
- DatagramChannel
- SocketChannel
- ServerSocketChannel

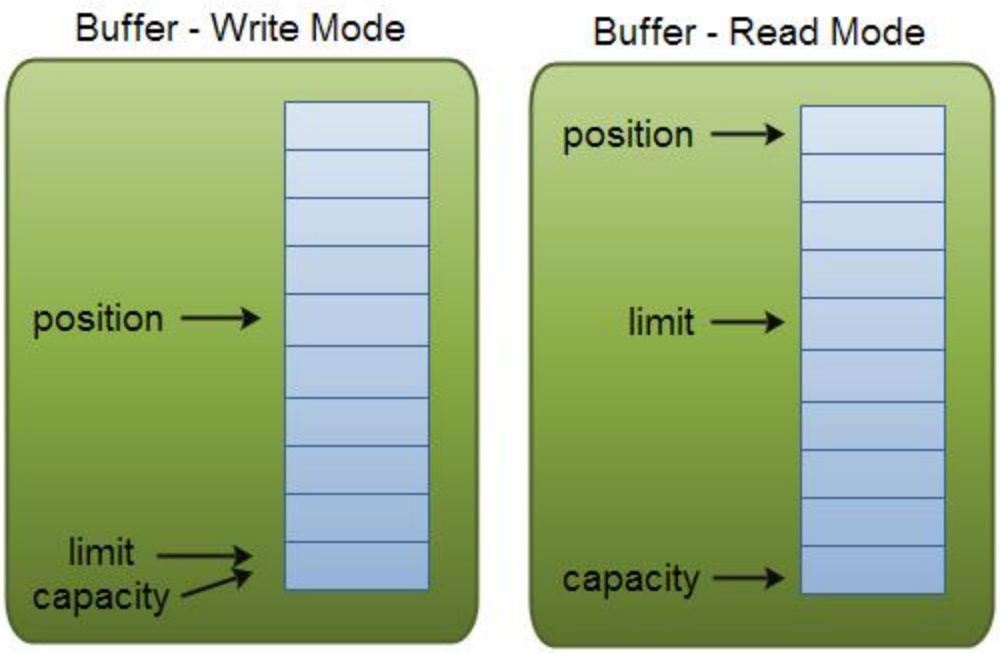
```
 RandomAccessFile aFile = new
RandomAccessFile("data/nio-data.txt", "rw");
    FileChannel inChannel = aFile.getChannel();

    ByteBuffer buf = ByteBuffer.allocate(48);

    int bytesRead = inChannel.read(buf);
    while (bytesRead != -1) {

      System.out.println("Read " + bytesRead);
      buf.flip();

      while(buf.hasRemaining()){
          System.out.print((char) buf.get());
      }

      buf.clear();
      bytesRead = inChannel.read(buf);
    }
    aFile.close();
```

Java NIO: Buffer

1. Write data into the Buffer
2. Call buffer.flip()
3. Read data out of the Buffer
4. Call buffer.clear() or buffer.compact()

```
RandomAccessFile aFile = new RandomAccessFile("data/nio-data.txt", "rw");
FileChannel inChannel = aFile.getChannel();

//create buffer with capacity of 48 bytes
ByteBuffer buf = ByteBuffer.allocate(48);

int bytesRead = inChannel.read(buf); //read into buffer.
while (bytesRead != -1) {

  buf.flip();  //make buffer ready for read

  while(buf.hasRemaining()){
      System.out.print((char) buf.get()); // read 1 byte at a time
  }

  buf.clear(); //make buffer ready for writing
  bytesRead = inChannel.read(buf);
}
aFile.close();
```

Buffer Capacity, Position and Limit



Buffer Types

- ByteBuffer
- MappedByteBuffer
- CharBuffer
- DoubleBuffer
- FloatBuffer
- IntBuffer
- LongBuffer
- ShortBuffer

Allocating a Buffer

```
ByteBuffer buf = ByteBuffer.allocate(48);

CharBuffer buf = CharBuffer.allocate(1024);
```

Writing Data to a Buffer

1. Write data from a Channel into a Buffer

```
int bytesRead = inChannel.read(buf); //read into buffer.
```

2. Write data into the Buffer yourself, via the buffer's put() methods.

```
buf.put(127);
```

flip() – switches from writing mode to reading

Reading Data from a Buffer

1. Read data from the buffer into a channel.

```
//read from buffer into channel.
int bytesWritten = inChannel.write(buf);
```

2. Read data from the buffer yourself, using one of the get() methods.

```
byte aByte = buf.get();
```

rewind() – set position back to 0

clear(), compact() – switches back to writing mode

mark(), reset() – mark given position

Buffer comparison

- equals, compareTo

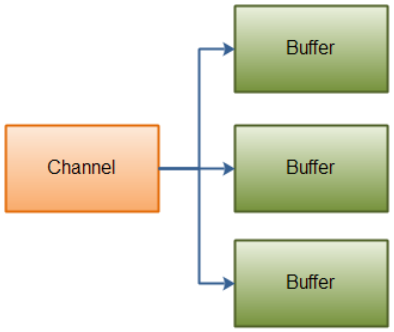
6th Lecture

- Java NIO(New IO) II
- JPA

Java NIO: Scatter / Gather

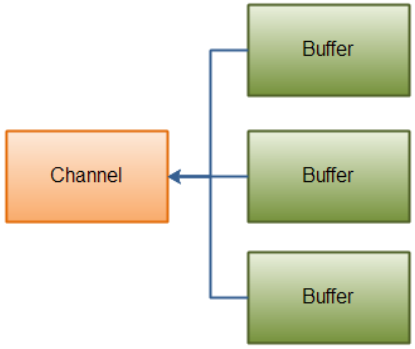
- Scattering Reads

```
ByteBuffer header = ByteBuffer.allocate(128);
ByteBuffer body   = ByteBuffer.allocate(1024);

ByteBuffer[] bufferArray = { header, body };

channel.read(bufferArray);
```



- Gathering Writes

```
ByteBuffer header = ByteBuffer.allocate(128);
ByteBuffer body   = ByteBuffer.allocate(1024);

//write data into buffers

ByteBuffer[] bufferArray = { header, body };

channel.write(bufferArray);
```

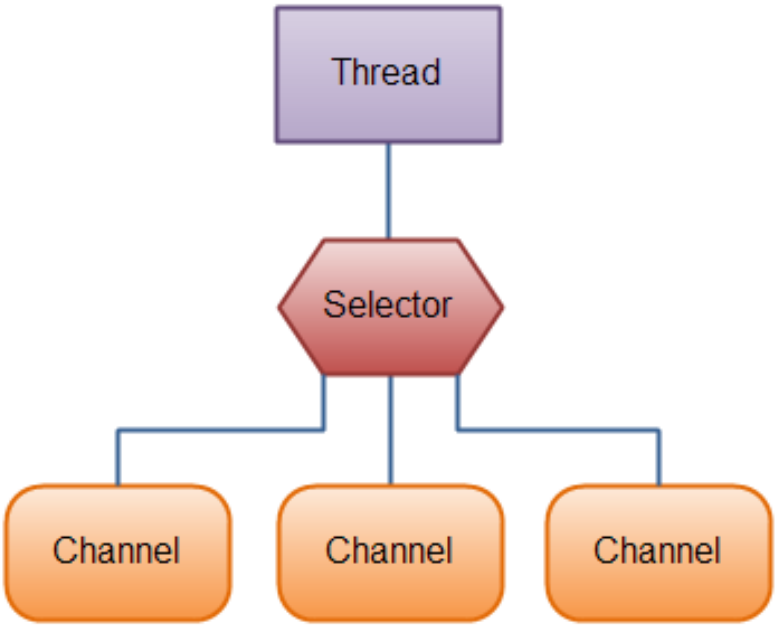


Java NIO: Channel to Channel Transfers

```
RandomAccessFile fromFile = new RandomAccessFile("fromFile.txt", "rw");
FileChannel      fromChannel = fromFile.getChannel();

RandomAccessFile toFile = new RandomAccessFile("toFile.txt", "rw");
FileChannel      toChannel = toFile.getChannel();

long position = 0;
long count    = fromChannel.size();

toChannel.transferFrom(fromChannel, position, count);
```

```
RandomAccessFile fromFile = new RandomAccessFile("fromFile.txt", "rw");
FileChannel      fromChannel = fromFile.getChannel();

RandomAccessFile toFile = new RandomAccessFile("toFile.txt", "rw");
FileChannel      toChannel = toFile.getChannel();

long position = 0;
long count    = fromChannel.size();

fromChannel.transferTo(position, count, toChannel);
```

Java NIO: Selector




```
Selector selector = Selector.open();

channel.configureBlocking(false);

SelectionKey key = channel.register(selector, SelectionKey.OP_READ);


while(true) {

  int readyChannels = selector.selectNow();

  if(readyChannels == 0) continue;


  Set<SelectionKey> selectedKeys = selector.selectedKeys();

  Iterator<SelectionKey> keyIterator = selectedKeys.iterator();

  while(keyIterator.hasNext()) {

    SelectionKey key = keyIterator.next();

    if(key.isAcceptable()) {
        // a connection was accepted by a ServerSocketChannel.

    } else if (key.isConnectable()) {
        // a connection was established with a remote server.

    } else if (key.isReadable()) {
        // a channel is ready for reading

    } else if (key.isWritable()) {
        // a channel is ready for writing
    }

    keyIterator.remove();
  }
}
```

Selector: Other methods

- wakeup() – allow leave select() even no channel is ready
- close()

String newData = "New String to write to file..." + System.currentTimeMillis(); ByteBuffer buf = ByteBuffer.allocate(48); buf.clear(); buf.put(newData.getBytes()); bu

Java NIO: FileChannel

- Opening

```
RandomAccessFile aFile     = new RandomAccessFile("data/nio-data.txt", "rw");
FileChannel      inChannel = aFile.getChannel();
```

- Reading and writing

```
String newData = "New String to write to file..."
+ System.currentTimeMillis();

ByteBuffer buf = ByteBuffer.allocate(48);
buf.clear();
buf.put(newData.getBytes());

buf.flip();

while(buf.hasRemaining()) {
    channel.write(buf);
}
```

FileChannel: other methods

- close()
- position(), position(int position)
- long size()
- truncate(long newSize)
- force()

Java NIO: SocketChannel

- Opening

```
SocketChannel socketChannel = SocketChannel.open();
socketChannel.connect(new InetSocketAddress("http://jenkov.com", 80))
```

- Reading, writing
- Non-blocking mode – connect, write, read

```
socketChannel.configureBlocking(false);
socketChannel.connect(new InetSocketAddress("http://jenkov.com", 80));

while(! socketChannel.finishConnect() ){
    //wait, or do something else...
}
```

Java NIO: ServerSocketChannel

```
ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();

serverSocketChannel.socket().bind(new InetSocketAddress(9999));

while(true){
    SocketChannel socketChannel =
            serverSocketChannel.accept();

    //do something with socketChannel...
}
serverSocketChannel.close();
```

ServerSocketChannel

```
ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();

serverSocketChannel.socket().bind(new InetSocketAddress(9999));
serverSocketChannel.configureBlocking(false);

while(true){
    SocketChannel socketChannel =
            serverSocketChannel.accept();

    if(socketChannel != null){
        //do something with socketChannel...
    }
}
```

Java NIO: DatagramChannel

- Opening

```
DatagramChannel channel = DatagramChannel.open();
channel.socket().bind(new InetSocketAddress(9999));
```

- Receiving Data

```
ByteBuffer buf = ByteBuffer.allocate(48);
buf.clear();

channel.receive(buf)
```

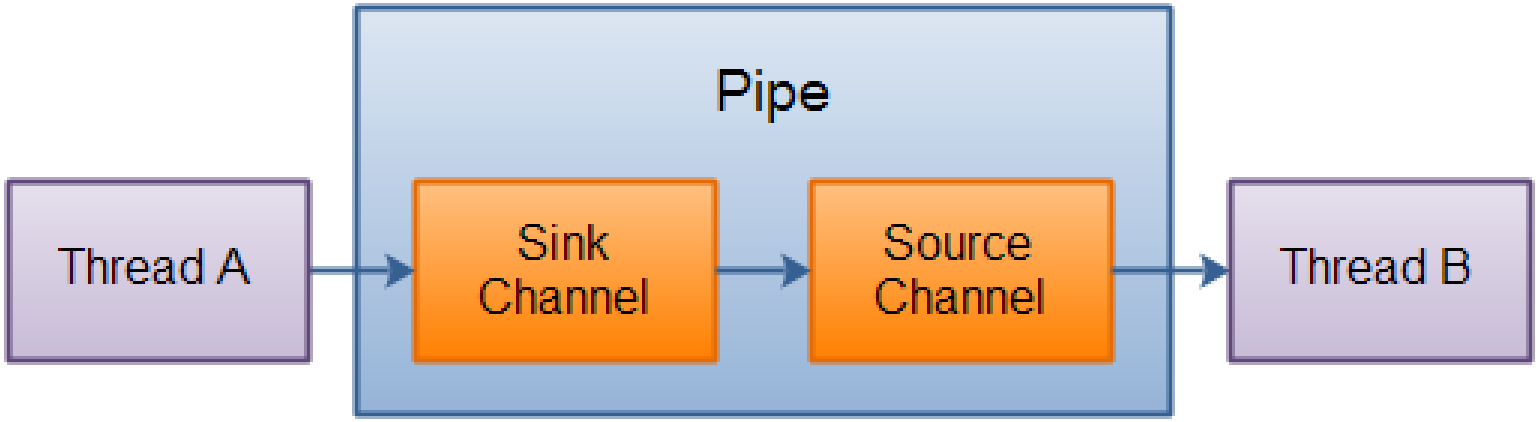
DatagramChannel: Sending Data

```
String newData = "New String to write to file..."
                    + System.currentTimeMillis();

ByteBuffer buf = ByteBuffer.allocate(48);
buf.clear();
buf.put(newData.getBytes());
buf.flip();

int bytesSent = channel.send(buf, new InetSocketAddress(“cs.vsb.cz", 80));
```

DatagramChannel: Connecting to a Specific Address

```
channel.connect(new InetSocketAddress(“cs.vsb.cz", 80));
…

int bytesRead = channel.read(buf);
…

int bytesWritten = channel.write(buf);
```

Java NIO: Pipe



Pipe: Writing

```
Pipe pipe = Pipe.open();

Pipe.SinkChannel sinkChannel = pipe.sink();

String newData = "New String to write to file..." + System.currentTimeMillis();

ByteBuffer buf = ByteBuffer.allocate(48);
buf.clear();
buf.put(newData.getBytes());

buf.flip();

while(buf.hasRemaining()) {
    sinkChannel.write(buf);
}
```

Pipe: Reading

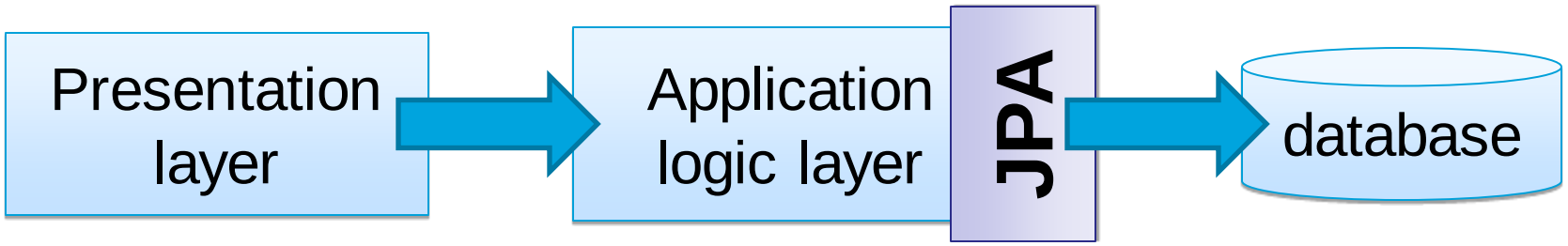
```
Pipe.SourceChannel sourceChannel = pipe.source();

ByteBuffer buf = ByteBuffer.allocate(48);


int bytesRead = inChannel.read(buf);
```

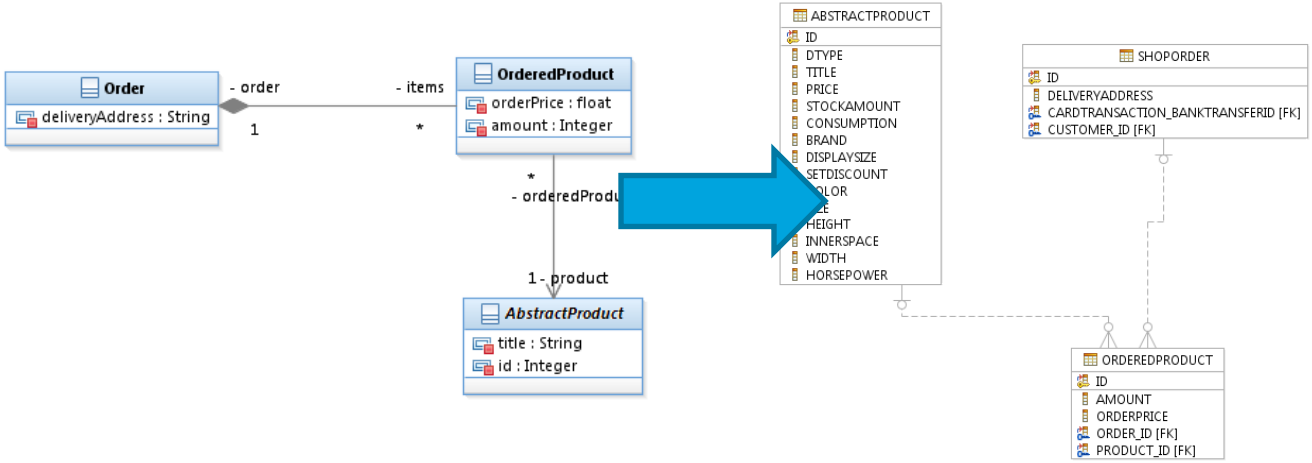
Java NIO vs IO

IO	NIO
Stream oriented	Buffer oriented
Blocking IO	Non blocking IO Selectors

JPA: overview

- API persistency using ORM
- Only interface – implementation should be connected.





JPA: Entity

- **Entity** – light-wight object from persistence object. Typicaly are related with database table. Each object is related to one record in the database table.
- **Persistent state of entity:** represented by instance variables and class properties. Mapping between database and properties is defined by annotations.

JPA – Entity class

- an annotation

`javax.persistence.Entity`

- Nonparametric public or protected constructor.
- Class nor methods nor instance variables are `final`
- If is entity used in remote EJB interface have to implemented interface `Serializable`
- Entity class can be descendant of entity class or non-entity class. Non-entity classes can be descendant of entity class.
- Persistence instance variables have to be declared as private, protected or package-private. They should be accessed through set and get methods.

JPA: example of Entity class

```
@Entity
@Table(name="ShopOrder")
public class Order {
 @Id
 @GeneratedValue(strategy=GenerationType.IDENTITY)
 private int id;
 @OneToOne
 private Transaction cardTransaction;
 @ManyToOne()
 private Customer customer;
 @OneToMany(mappedBy="order")
 private Set<OrderedProduct> items;
 private String deliveryAddress;
 …
}
```

JPA: persistence properties, instance variables

- Instance variables – persistence provider access directly to them
- Properties – Persistence access properties using get, set method
  - Can be used: Collection, Set, List, Map even generic versions
- override `equals()` `hashcode()`
- Types:
  - Java primitive data types
  - java.lang.String,
  - other serializable types (boxed classes, java.math.BigInteger, java.math.BigDecimal, java.util.Date, java.util.Calendar, java.sql.Date, java.sql.Time, java.sql.TimeStamp, user serializable types, byte[], Byte[], char[], Character[], enum types, other entities,

JPA: primary key

- Every entity should contain own key
- `javax.persistence.Id`
- Composite Primary Key
  - Have to exisit class which define composite key
  - `javax.persistence.EmbeddedId`
  - `javax.persistence.IdClass`
  - Have to be composed from types:
    - Java Primitive data types (and coresponding embedded classes)
    - java.lang.String
    - java.util.Date (DATE), java.sql.Date
- Float numbers should not be used.

JPA: class for primary key

- Has to:
  - be public
  - Have public or protected instance variables but get/set public methods have to exists
  - Public nonparametric constructor
  - Override `equals(), hashcode()`
  - Implements `Serializable`
  - Be mapped on a couple instance variables or properties of entity
  - Names of variables/properties should be in relation with names in entity

JPA: example of primary composite key

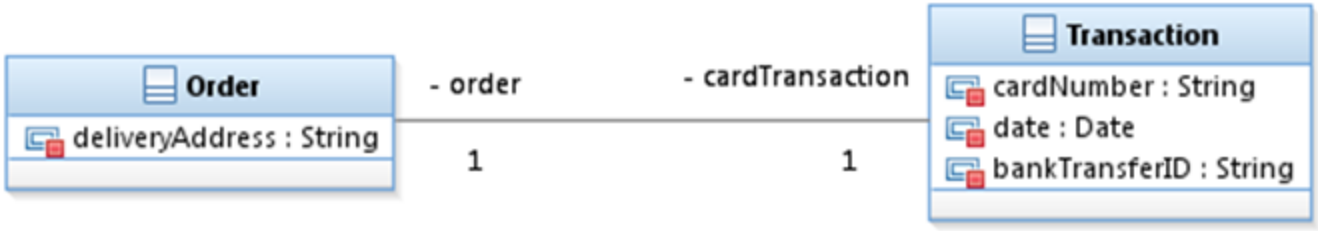
```
@Entity
@IdClass(AccountId.class)
public class Account {
    @Id
    private String accountNumber;

    @Id
    private String accountType;

    // other fields, getters and setters
}
```

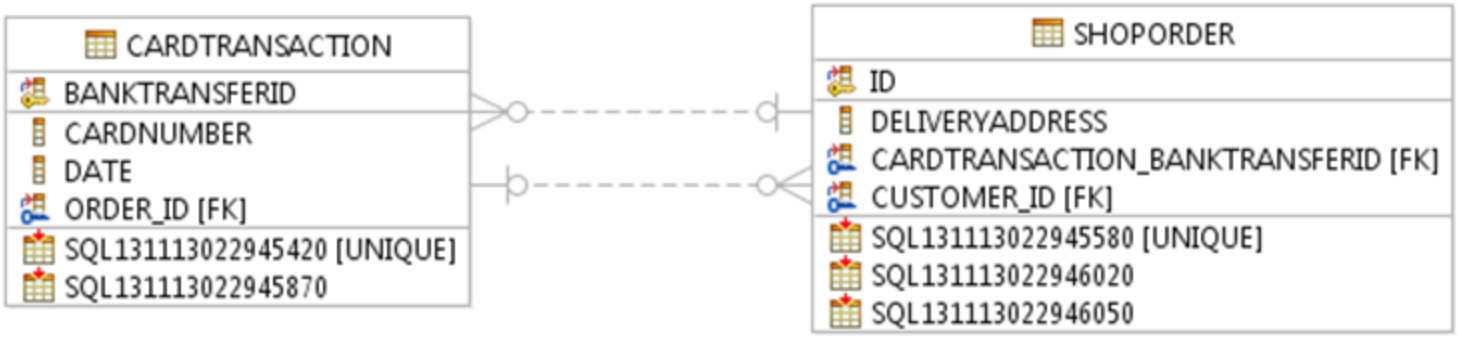
```
public class AccountId implements Serializable {
    private String accountNumber;

    private String accountType;

    // default constructor

    public AccountId(String accountNumber, String
accountType) {
        this.accountNumber = accountNumber;
        this.accountType = accountType;
    }

    // equals() and hashCode()
}
```

JPA – relation 1-1

```
@Entity
public class Order {
@OneToOne
private Transaction cardTransaction;
…
```

```
@Entity
public class Transaction {
@OneToOne
private Order order;
```

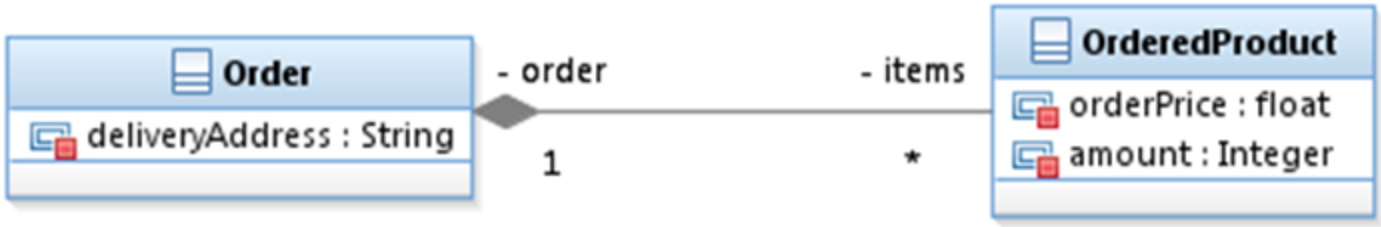


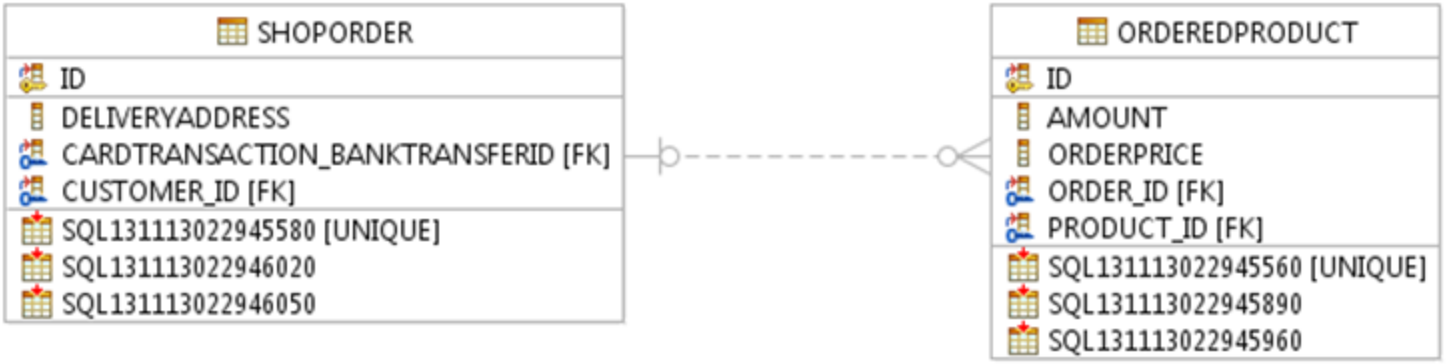


JPA – relation 1-N

```
@Entity
public class Order {
@OneToMany(mappedBy="order")
private Set<OrderedProduct> items;
…
```

```
@Entity
public class OrderedProduct {
@ManyToOne
private Order order;
```

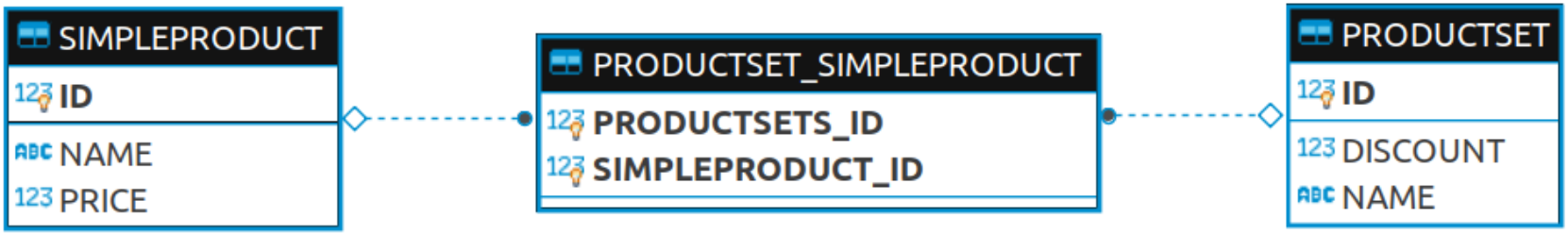




JPA – relation M-N

```
@Entity
public class SimpleProduct extends
AbstractProduct {
@ManyToMany(mappedBy="simpleProduct")
private List<ProductSet> productSets;
}
```

```
@Entity
public class ProductSet extends
AbstractProduct @ManyToMany
private List<SimpleProduct> simpleProduct;
private float setDiscount;
}
```



JPA - inheritence

- Entity can extend non-entity or abstract class

```
@Entity
public abstract class Employee {
	@Id
	protected Integer employeeId;
}
...
@Entity
public class FullTimeEmployee extends Employee {
	protected Integer salary;
}
…
@Entity
public class PartTimeEmployee extends Employee {
	protected Float hourlyWage;
}
```

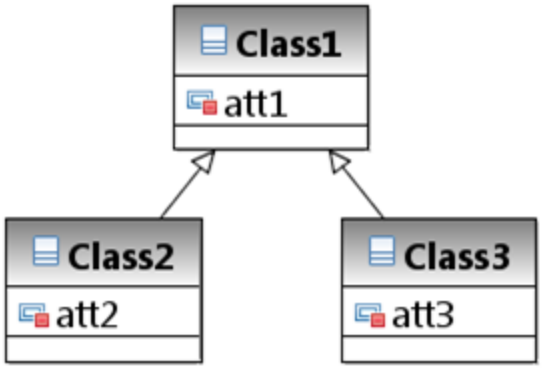
JPA – inheritance mapping strategy

- One table on a class hierarchy
- One table for a particular class
- Join strategy

```
public enum InheritanceType {
          SINGLE_TABLE,
          JOINED,
          TABLE_PER_CLASS
};
…
@Inheritance(strategy=JOINED)
```

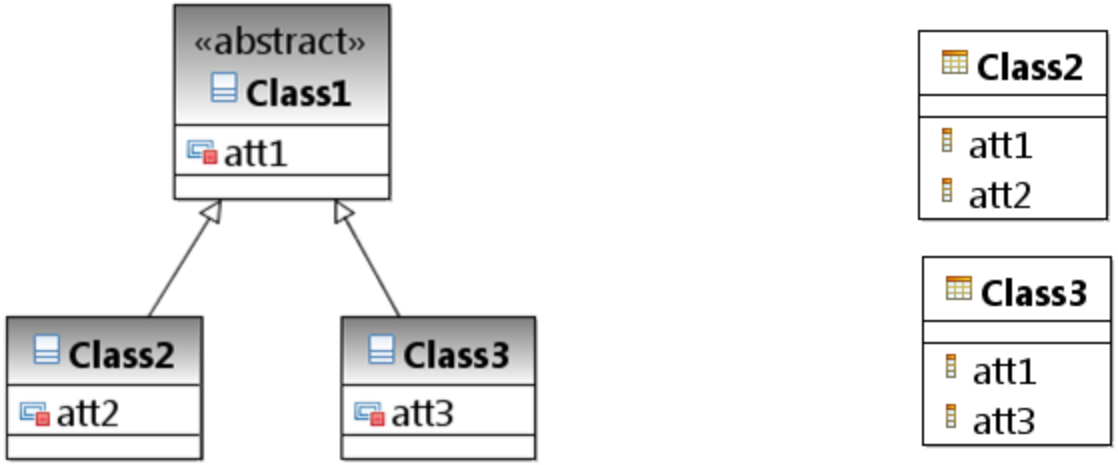
JPA: SINGLE_TABLE

```
@Inheritance(strategy=SINGLE_TABLE)

@DiscriminatorColumn(
String name
DiscriminatorType discriminatorType
String columnDefinition
String length)
public enum DiscriminatorType {
        STRING,
        CHAR,
        INTEGER
};
@DiscriminatorValue
```



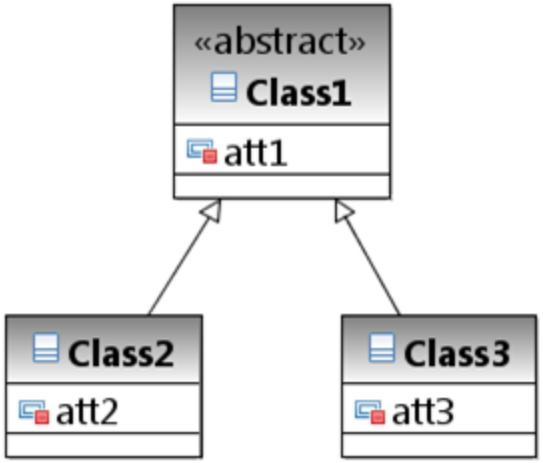
JPA: One table for particular entity

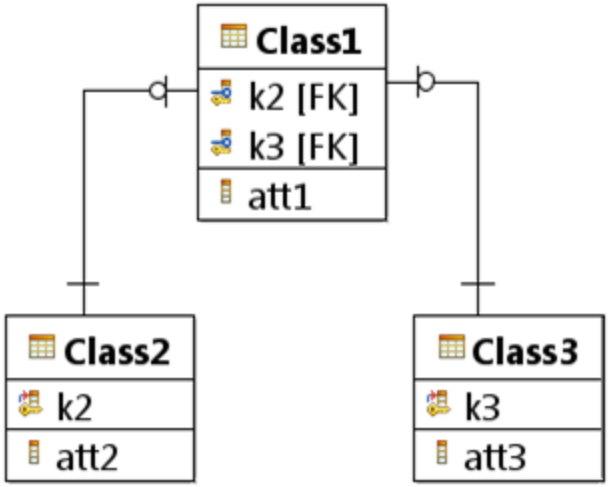
@Inheritance(strategy=TABLE_PER_CLASS)



JPA: Join strategy

```
@Inheritance(strategy=JOINED)
```





JPA: MappedSuperclass

```
@MappedSuperclass
public class Person {
	@Column(length=50)
	private String name;
	@Column(length=50)
	private String surename;
	@Column(length=50)
	private String email;
	@Column(length=50)
	private String password;
}
```

```
@Entity
public class Customer extends Person {
	@Id
	@GeneratedValue(strategy=GenerationType.IDENTIT
	Y)
	private int id;
	@OneToMany(mappedBy="customer")
	private Set<Order> orders;
```

```
@Entity
public class Employee extends Person {
	@Id
	@Column(length=50)
	private String login;
	private float sallary;
	@Column(length=50)
	private String depsrtment;
}
```

JPA: entity management

- Persistent context: set of entities existing in a particular datastore
- EntityManager:
  - Creates, deletes, finds, exetues queries

JPA: Application managed EntityManager

```
EntityManagerFactory emf = // fetched from somewhere
EntityManager em = emf.createEntityManager();
// use it in the current thread
```

JPA: find entity

```
@PersistenceContext
EntityManager em;

public void enterOrder(int custID, Order newOrder) {
            Customer cust = em.find(Customer.class, custID);
            cust.getOrders().add(newOrder);
            newOrder.setCustomer(cust);
}
```

JPA: entity lifecycle

- New
- Managed
- Detached
- Removed

```
EntityManager em;
...
public LineItem createLineItem(Order order, Product product, int quantity) {
            LineItem li = new LineItem(order, product, quantity);
            order.getLineItems().add(li);
            em.persist(li);
            return li;
}

em.remove(order);
em.flush();
```

JPA: queries

```
public List findWithName(String name) {
          return em.createQuery(
          "SELECT c FROM Customer c WHERE c.name LIKE :custName")
          .setParameter("custName", name)
          .setMaxResults(10)
          .getResultList();
}

…
query.setFirstResult(100)
```

JPA: named queries

```
@NamedQuery(//class annotation - entity
        name="findAllCustomersWithName",
        query="SELECT c FROM Customer c WHERE c.name LIKE :custName"
)

EntityManager em;
...
customers = em.createNamedQuery("findAllCustomersWithName")
        .setParameter("custName", "Smith")
        .getResultList();
```

JPA: parameters in queris

- Named

```
return em.createQuery(
        "SELECT c FROM Customer c WHERE c.name LIKE :custName")
        .setParameter("custName", name)
        .getResultList();
```

- Numbered

```
return em.createQuery(
        “SELECT c FROM Customer c WHERE c.name LIKE ?1”)
        .setParameter(1, name)
        .getResultList();
```

JPA: Persistence Units

- Package containing all entity class mapped on one datastore
- must contain file persistence.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
      <persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
      http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
         <persistence-unit name="Slaids">
                  <provider>org.hibernate.ejb.HibernatePersistence</provider>
                  <jta-data-source>java:/jdbc/slaids</jta-data-source>
                  <properties>
                           <property name="javax.persistence.schema-generation.database.action" value="create"/>
                           <property name="hibernate.hbm2ddl.auto" value="create"/>
                           <property name="hibernate.dialect" value="org.hibernate.dialect.DerbyTenSevenDialect"/>
                  </properties>
      </persistence-unit>
</persistence>
```

JPA – Query Language

- Select Statements
  - SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY
- Update, Delete Statement
  - UPDATE Player p SET p.status = ’inactive’ WHERE p.lastPlayed < :inactiveThresholdDate
  - DELETE FROM Player p WHERE p.status = ’inactive’ AND p.teams IS EMPTY

JPA: examples of queries

- SELECT p FROM Player AS p
- SELECT DISTINCT p FROM Player AS p WHERE p.position = ?1
- SELECT DISTINCT t FROM Player AS p JOIN p.teams AS t
- SELECT DISTINCT p FROM Player AS p WHERE p.team IS NOT EMPTY
- SELECT t FROM Team AS t JOIN t.league AS l WHERE l.sport = ’soccer’ OR l.sport =’football’
- SELECT DISTINCT p FROM Player AS p JOIN p.teams AS t WHERE t.city = :city
- SELECT DISTINCT p FROM Player AS p JOIN p.teams AS t WHERE t.league.sport = :sport

JPA: LIKE in query

- SELECT p FROM Player p WHERE p.name LIKE ’Mich%’
- _ - any one character
- % - zero or many any characters
- ESCAPE – defines escape character

- LIKE ’_%’ ESCAPE ’\’
- NOT LIKE

JPA: IS EMPTY, NULL in queries

- SELECT t FROM Team t WHERE t.league IS NULL
- SELECT t FROM Team t WHERE t.league IS NOT NULL
- Cannot use WHERE t.league = NULL

- SELECT p FROM Player p WHERE p.teams IS EMPTY
- SELECT p FROM Player p WHERE p.teams IS NOT EMPTY

JPA – BETWEEN, IN in queries

- SELECT DISTINCT p FROM Player p WHERE p.salary BETWEEN :lowerSalary AND :higherSalary
- p.salary >= :lowerSalary AND p.salary <= :higherSalary

- o.country IN (’UK’, ’US’, ’France’)

JPA Criteria[1]

- It enables us to write queries without doing raw SQL
- gives us some object-oriented control over the queries,

[1] https://www.baeldung.com/hibernate-criteria-queries

JPA Criteria - example

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Item> cr = cb.createQuery(Item.class);
Root<Item> root = cr.from(Item.class);
cr.select(root);
Query<Item> query = session.createQuery(cr);
List<Item> results = query.getResultList();
```

JPA criteria – using expression

```
cr.select(root).where(cb.gt(root.get("itemPrice"), 1000));

cr.select(root).where(cb.like(root.get("itemName"), "%chair%"));

cr.select(root).where(cb.between(root.get("itemPrice"), 100, 200));
```

JPA criteria - predicate chaining

```
Predicate greaterThanPrice = cb.gt(root.get("itemPrice"), 1000);

Predicate chairItems = cb.like(root.get("itemName"), "Chair%");

cr.select(root).where(cb.or(greaterThanPrice, chairItems));
```

JPA criteria – using metamodel[1]

```java
@Entity
@Table(name = "students")
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "grad_year")
    private int gradYear;

    // standard getters and setters
}
```

```java
@Generated(value = "org.hibernate.jpamodelgen.JPAMetaModelEntityProcessor")
@StaticMetamodel(Student.class)
public abstract class Student_ {

    public static volatile SingularAttribute<Student, String> firstName;
    public static volatile SingularAttribute<Student, String> lastName;
    public static volatile SingularAttribute<Student, Integer> id;
    public static volatile SingularAttribute<Student, Integer> gradYear;

    public static final String FIRST_NAME = "firstName";
    public static final String LAST_NAME = "lastName";
    public static final String ID = "id";
    public static final String GRAD_YEAR = "gradYear";
}
```

```java
//session set-up code
CriteriaBuilder cb = session.getCriteriaBuilder();
CriteriaQuery<Student> criteriaQuery = cb.createQuery(Student.class);
```

[1]https://www.baeldung.com/hibernate-criteria-queries-metamodel

7h lecture

- Serialization
- Java Reflection

Serialization: Reference

https://www.oracle.com/technical-resources/articles/java/serializationapi.html

Serialization

- Rule #1 – Make class *Serializable*
- Rule #2 – Nonserializable fields mark *transient*

Serialization: Customize the Default Protocol

```
    private void writeObject(ObjectOutputStream out) throws IOException;
    private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException;
```

```
import java.io.Serializable;
public class PersistentAnimation implements Serializable, Runnable
{
          transient private Thread animator;
          private int animationSpeed;
          private void writeObject(ObjectOutputStream out) throws IOException
          {
                    out.defaultWriteObject();
          }

          private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException
          {
                    // our "pseudo-constructor"
                    in.defaultReadObject();
                    // now we are a "live" object again, so let's run rebuild and start
                    startAnimation();
}
```

Serialization: the Externalizable Interface

```
public void writeExternal(ObjectOutput out) throws IOException;

public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException;
```

Serialization: Caching Objects in the Stream

- ObjectOutputStream.reset()

```
ObjectOutputStream out = new ObjectOutputStream(...);
MyObject obj = new MyObject(); // must be Serializable
obj.setState(100);
out.writeObject(obj); // saves object with state = 100
obj.setState(200);
out.writeObject(obj); // does not save new object state
```

Serialization: Version Control

```
static final long serialVersionUID = 1027553947283749 5L;
```

Serialization: Performance Considerations

- If speed is the primary consideration for your application, you may want to consider building a custom protocol.

Java Reflection: Reference

http://tutorials.jenkov.com/java-reflection/index.html

Java Reflection

```
Method[] methods = MyObject.class.getMethods();

for(Method method : methods){
    System.out.println("method = " + method.getName());
}
```

Java Reflection: Class object

```
Class myObjectClass = MyObject.class

myObjectClass = obj.getClass();

myObjectClass = Class.forName("some.MyClass");
```

Java Reflection: Class Name

```
String className = aClass.getName();

String simpleClassName = aClass.getSimpleName();
```

Java Reflection: Class modifiers

```
Class  aClass = ... //obtain Class object. See prev. section
int modifiers = aClass.getModifiers();
```

```
Modifier.isAbstract(int modifiers)
Modifier.isFinal(int modifiers)
Modifier.isInterface(int modifiers)
Modifier.isNative(int modifiers)
Modifier.isPrivate(int modifiers)
Modifier.isProtected(int modifiers)
Modifier.isPublic(int modifiers)
Modifier.isStatic(int modifiers)
Modifier.isStrict(int modifiers)
Modifier.isSynchronized(int modifiers)
Modifier.isTransient(int modifiers)
Modifier.isVolatile(int modifiers)
```

Java Reflection: Other class features

```
Class  aClass = ... //obtain Class object. See prev. section
Package package = aClass.getPackage();


Class superclass = aClass.getSuperclass();


Class  aClass = ... //obtain Class object. See prev. section
Class[] interfaces = aClass.getInterfaces();
```

Java Reflection: Constructors

```
  Constructor[] constructors = aClass.getConstructors();

…
  Class aClass = ...//obtain class object
  Constructor constructor =
        aClass.getConstructor(new Class[]{String.class});


…

  Constructor constructor = ... // obtain constructor - see above
  Class[] parameterTypes = constructor.getParameterTypes();

…


//get constructor that takes a String as argument
Constructor constructor = MyObject.class.getConstructor(String.class);

MyObject myObject = (MyObject)
        constructor.newInstance("constructor-arg1");
```

Java Reflection: Methods

```
Class aClass = ...//obtain class object
Method[] methods = aClass.getMethods();

…

Class  aClass = ...//obtain class object
Method method =
    aClass.getMethod("doSomething", new Class[]{String.class});

…

Class  aClass = ...//obtain class object
Method method =
    aClass.getMethod("doSomething", null);

…

Method method = ... // obtain method - see above
Class[] parameterTypes = method.getParameterTypes();
Class returnType = method.getReturnType();

…

//get method that takes a String as argument
Method method = MyObject.class.getMethod("doSomething", String.class);

Object returnValue = method.invoke(null, "parameter-value1");
```

Java Reflection: Access private methods

```
public class PrivateObject {

  private String privateString = null;

  public PrivateObject(String privateString) {
    this.privateString = privateString;
  }
}
```

```
PrivateObject privateObject = new PrivateObject("The Private Value");

Field privateStringField = PrivateObject.class.
            getDeclaredField("privateString");

privateStringField.setAccessible(true);

String fieldValue = (String) privateStringField.get(privateObject);
System.out.println("fieldValue = " + fieldValue);
```

Java Reflection: Fields

```
public class MyObject{
  public String someField = null;

}
```

```
Class  aClass = MyObject.class;
Field[] fields = aClass.getFields();
Field field = aClass.getField("someField");

…

String fieldName = field.getName();
Object fieldType = field.getType();

…

MyObject objectInstance = new MyObject();

Object value = field.get(objectInstance);

field.set(objetInstance, value);
```

Java Reflection: Annotations

```
@MyAnnotation(name="someName",  value = "Hello World")
public class TheClass {
}
```

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)

public @interface MyAnnotation {
    public String name();
    public String value();
}
```

Java Reflection: Class Annotations

```
Class aClass = TheClass.class;
Annotation[] annotations = aClass.getAnnotations();

for(Annotation annotation : annotations){
    if(annotation instanceof MyAnnotation){
        MyAnnotation myAnnotation = (MyAnnotation) annotation;
        System.out.println("name: " + myAnnotation.name());
        System.out.println("value: " + myAnnotation.value());
    }
}
```

```
Class aClass = TheClass.class;
Annotation annotation = aClass.getAnnotation(MyAnnotation.class);

if(annotation instanceof MyAnnotation){
    MyAnnotation myAnnotation = (MyAnnotation) annotation;
    System.out.println("name: " + myAnnotation.name());
    System.out.println("value: " + myAnnotation.value());
}
```

Java Reflection: Methods Annotations

```
public class TheClass {
  @MyAnnotation(name="someName",  value = "Hello World")
  public void doSomething(){}
}
```

```
Method method = ... //obtain method object
Annotation[] annotations = method.getDeclaredAnnotations();

for(Annotation annotation : annotations){
    if(annotation instanceof MyAnnotation){
        MyAnnotation myAnnotation = (MyAnnotation) annotation;
        System.out.println("name: " + myAnnotation.name());
        System.out.println("value: " + myAnnotation.value());
    }
}
```

```
Method method = ... // obtain method object
Annotation annotation = method.getAnnotation(MyAnnotation.class);

if(annotation instanceof MyAnnotation){
    MyAnnotation myAnnotation = (MyAnnotation) annotation;
    System.out.println("name: " + myAnnotation.name());
    System.out.println("value: " + myAnnotation.value());
}
```

Java Reflection: Parameter Annotations

```
public class TheClass {
  public static void doSomethingElse(
        @MyAnnotation(name="aName", value="aValue") String parameter){
  }
}
```

Java Reflection: Field Annotations

```
public class TheClass {

  @MyAnnotation(name="someName",  value = "Hello World")
  public String myField = null;
}
```

Java Reflection: Arrays

```
int[] intArray = (int[]) Array.newInstance(int.class, 3);
…
int[] intArray = (int[]) Array.newInstance(int.class, 3);

Array.set(intArray, 0, 123);
Array.set(intArray, 1, 456);
Array.set(intArray, 2, 789);

System.out.println("intArray[0] = " + Array.get(intArray, 0));
System.out.println("intArray[1] = " + Array.get(intArray, 1));
System.out.println("intArray[2] = " + Array.get(intArray, 2));
…
Class stringArrayClass = String[].class;

Class intArray = Class.forName("[I");

Class stringArrayClass = Class.forName("[Ljava.lang.String;");
…
Class stringArrayClass = Array.newInstance(String.class, 0).getClass();
System.out.println("is array: " + stringArrayClass.isArray());
…
String[] strings = new String[3];
Class stringArrayClass = strings.getClass();
Class stringArrayComponentType = stringArrayClass.getComponentType();
System.out.println(stringArrayComponentType);
```

Java Reflection: Dynamic Proxies

1. The ClassLoader that is to "load" the dynamic proxy class.
2. An array of interfaces to implement.
3. An InvocationHandler to forward all methods calls on the proxy to.

```
InvocationHandler handler = new MyInvocationHandler();
MyInterface proxy = (MyInterface) Proxy.newProxyInstance(
                            MyInterface.class.getClassLoader(),
                            new Class[] { MyInterface.class },
                            handler);
```

Java Reflection: Invocation handler

```
public interface InvocationHandler{
  Object invoke(Object proxy, Method method, Object[] args)
         throws Throwable;
}



public class MyInvocationHandler implements InvocationHandler{

  public Object invoke(Object proxy, Method method, Object[] args)
  throws Throwable {
    //do something "dynamic"
  }
}
```

Java Reflection: Dynamic Proxies Known Use Cases

- Database Connection and Transaction Management
- Dynamic Mock Objects for Unit Testing
- Adaptation of DI Container to Custom Factory Interfaces
- AOP-like Method Interception

Java Reflection: Dynamic Class Loading

```
public class MainClass {

  public static void main(String[] args){

    ClassLoader classLoader = MainClass.class.getClassLoader();

    try {
        Class aClass = classLoader.loadClass("com.jenkov.MyClass");
        System.out.println("aClass.getName() = " + aClass.getName());
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }

}
```

Java Reflection: Own ClassLoader

```
public class MyClassLoader extends ClassLoader{

    public MyClassLoader(ClassLoader parent) {
        super(parent);
    }

    public Class loadClass(String name) throws ClassNotFoundException {
        if(!"reflection.MyObject".equals(name))
                return super.loadClass(name);

        try {
            String url = "file:C:/data/projects/tutorials/web/WEB-INF/" +
                            "classes/reflection/MyObject.class";
            URL myUrl = new URL(url);
            URLConnection connection = myUrl.openConnection();
            InputStream input = connection.getInputStream();
            ByteArrayOutputStream buffer = new ByteArrayOutputStream();
            int data = input.read();

            while(data != -1){
                buffer.write(data);
                data = input.read();
            }

            input.close();

            byte[] classData = buffer.toByteArray();

            return defineClass("reflection.MyObject",
                    classData, 0, classData.length);

        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

        return null;
    }

}
```

Java Reflection: Class Reloading

```
public static void main(String[] args) throws
    ClassNotFoundException,
    IllegalAccessException,
    InstantiationException {

    ClassLoader parentClassLoader = MyClassLoader.class.getClassLoader();
    MyClassLoader classLoader = new MyClassLoader(parentClassLoader);
    Class myObjectClass = classLoader.loadClass("reflection.MyObject");

    AnInterface2       object1 =
            (AnInterface2) myObjectClass.newInstance();

    MyObjectSuperClass object2 =
            (MyObjectSuperClass) myObjectClass.newInstance();

    //create new class loader so classes can be reloaded.
    classLoader = new MyClassLoader(parentClassLoader);
    myObjectClass = classLoader.loadClass("reflection.MyObject");

    object1 = (AnInterface2)       myObjectClass.newInstance();
    object2 = (MyObjectSuperClass) myObjectClass.newInstance();

}
```

8th Lecture

- REST
- HTTP-based RESTful API
- Quarkus

REST – REpresentational State Transfer

- Software architectural style
- Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)". Architectural Styles and the Design of Network-based Software Architectures (Ph.D.). University of California, Irvine.

REST: Architectural constraints

- Client-server architecture
- Statelessness
- Cacheability
- Layered systém
- Code on demand
- Uniform interface

REST: Client-Server

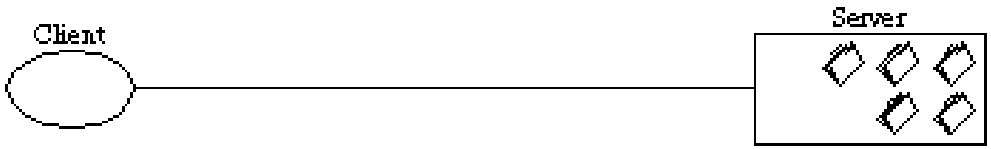


Figure 5-2. Client-Server

https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

REST: Stateless

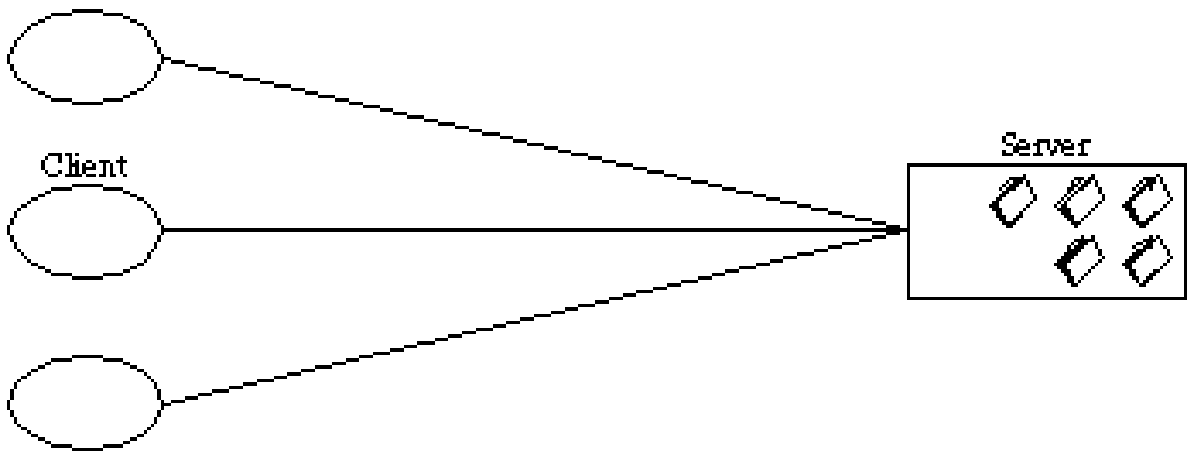


Figure 5-3. Client-Stateless-Server

https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

REST: Cache

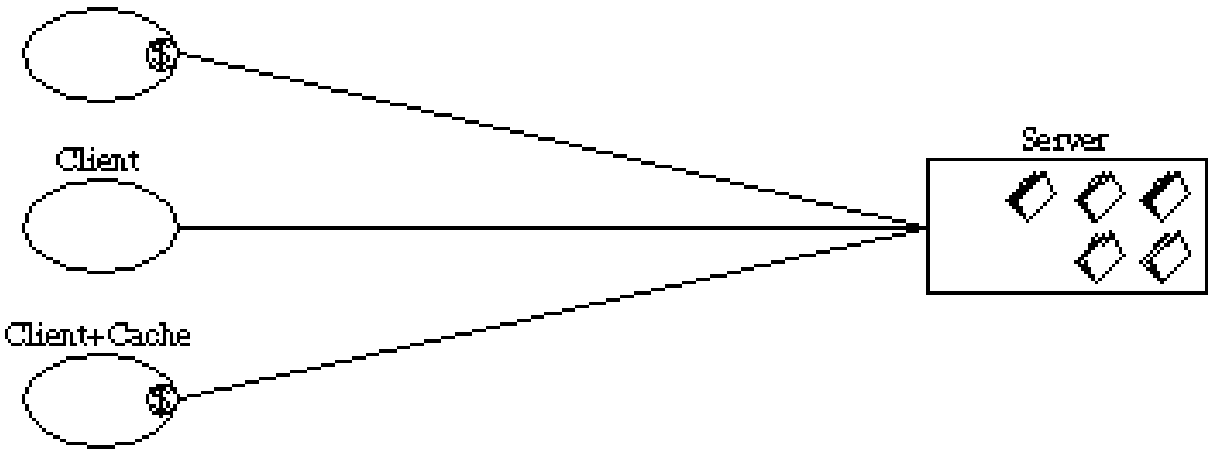


Figure 5-4. Client-Cache-Stateless-Server

https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

REST: Uniform Interface

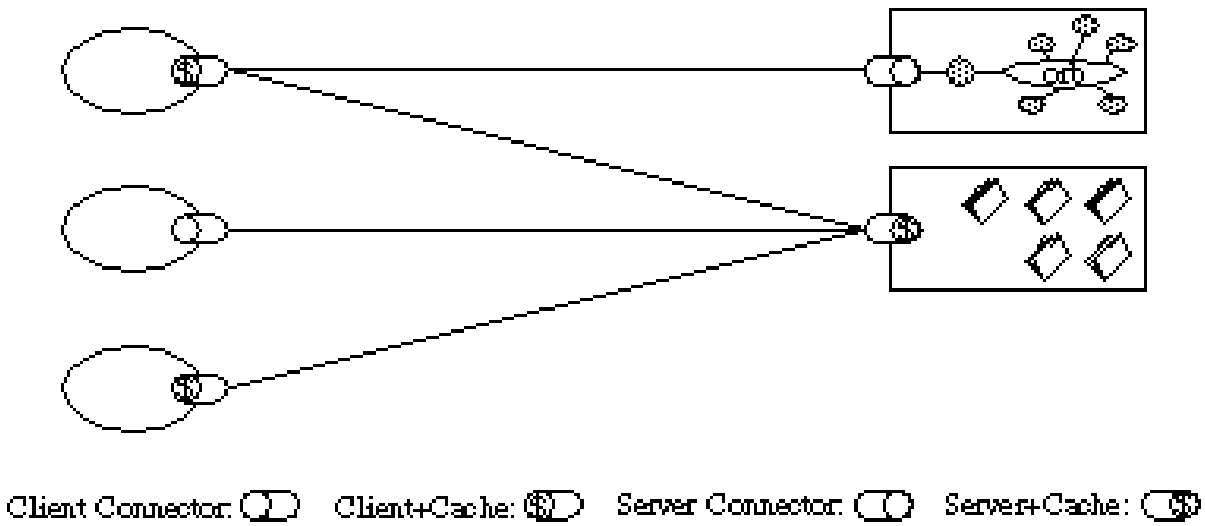


Figure 5-6. Uniform-Client-Cache-Stateless-Server

https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

REST: Uniform interface

- Resource identification in requests
- Resource manipulation through representations
- Self-descriptive messages
- Hypermedia as the engine of application state (HATEOAS)

REST: Layered System

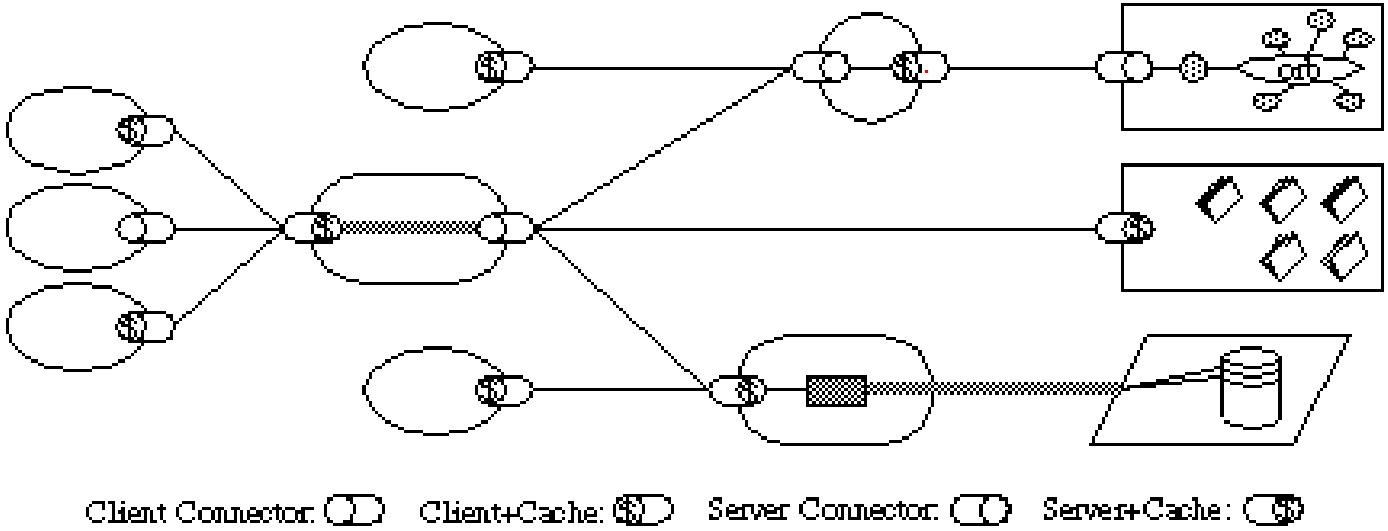


Figure 5-7. Uniform-Layered-Client-Cache-Stateless-Server

https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

REST: Code-On-Demand

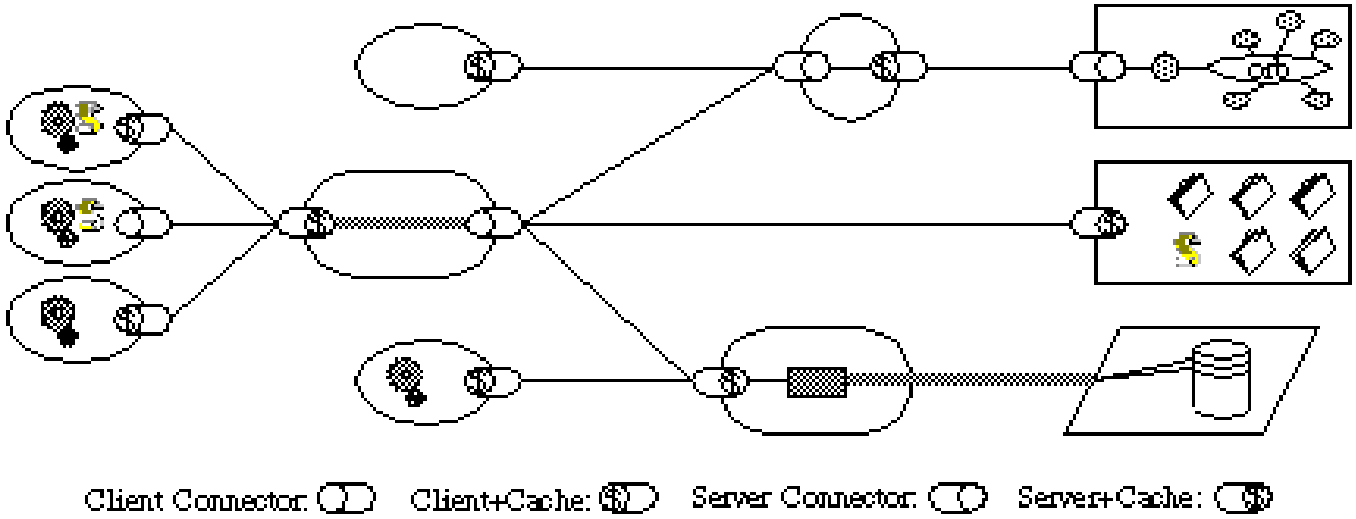


Figure 5-8. REST

https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

HTTP-based RESTful API

- Web service API
- REST architectural constraints
- Protocol HTTP – most common implementation:
  - URI
  - HTTP methods
  - media type

Semantic of HTTP methods

HTTP method	Description	CRUD	
GET	Get a representation of the target resource's state.	Fetch all or any resource	GET /user/ GET /user/1
POST	Let the target resource process the representation enclosed in the reques.	Create a Resources	POST /user?name=user17age=20
PUT	Set the target resurce's state tot the state defined by the representation enclosed in the request.	Update a Resource	PUT /user/1?name=changed-name
DELETE	Delete the target resource’s state.	Delete a Resource	DELETE /user/1
HEAD	Fetch metainfo		HEAD /user
OPTIONS	Fetch all verbs allowed		OPTIONS /user

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

HTTP Headers

Headers	Example
Auth: <session-token>	Auth: 1155dassdasd5-asd5666asd-asdas
Accept: <media Type>	Accept:application/json
Content-Type: <ct>	Content-Type: text/html; charset=UTF-8
Allow:<methods>	Allow: GET, POST, HEAD

HTTP Status Codes

- 1XX – Informational
- 2XX – Sucess
- 3XX – Redirection
- 4XX – Client Error
- 5XX – Server Error

Status Codes	
200 – OK	Successful Return for sync call
307 – Temporarily Moved	Redirection
400 – Bad Request	Invalid URI, header, request param
401 – Un authorized	User not authorized for operation
403 – Forbidden	User not allowed to update
404 – Not Found	URI Path not available
405 – Method not allowed	Method not valid for the path
500 – Internal Server Error	Server Errors
503 – Service unavailable	Server not accessible

JAX-RS (Java API for RESTful Web Services or Jakarta RESTful Web Services)

```
@ApplicationPath("/api")
public class RestApplication extends Application {
}
```

```
@Path("/notifications")
public class NotificationsResource {
  @GET
  @Path("/ping")
  public Response ping() {
    return Response.ok().entity("Service online").build();
  }

  @GET
  @Path("/{id}")
  @Produces(MediaType.APPLICATION_JSON)
  public Response getNotification(@PathParam("id") int id) {
    return Response.ok()
     .entity(new Notification(id, "john", "test notification"))
     .build();
  }

  @POST
  @Path("/")
  @Consumes(MediaType.APPLICATION_JSON)
  @Produces(MediaType.APPLICATION_JSON)
  public Response postNotification(Notification notification) {
    return Response.status(201).entity(notification).build();
  }
}
```

JAX-RS: Path and Query parameters

```
@Path("/")
public class DatasetRegisterServiceEndpoint {

   public static final String UUID = "uuid";
   ...


   @Path("datasets"
       +"/{" + UUID + "}"
        +"/{" + R_X_PARAM + "}"
        +"/{" + R_Y_PARAM + "}"
        +"/{" + R_Z_PARAM +    "}"
        +"/{" + VERSION_PARAM + "}"
        +"/{" + MODE_PARAM + "}")
   @GET
   public Response start(@PathParam(UUID) String uuid,
      @PathParam(R_X_PARAM) int rX, @PathParam(R_Y_PARAM) int rY,
      @PathParam(R_Z_PARAM) int rZ, @PathParam(VERSION_PARAM) String version,
      @SuppressWarnings("unused") @PathParam(MODE_PARAM) String mode,
      @QueryParam(TIMEOUT_PARAM) Long timeout)
        {
     //EXAMPLE Query: GET /datasets/445666-5555644-555555/1/1/1/latest/write?timeout=10000
```

JAX-RS: Build HTTP response

```
@GET
public Response start(@PathParam(UUID) String uuid,
   @PathParam(R_X_PARAM) int rX, @PathParam(R_Y_PARAM) int rY,
   @PathParam(R_Z_PARAM) int rZ, @PathParam(VERSION_PARAM) String version,
   @SuppressWarnings("unused") @PathParam(MODE_PARAM) String mode,
   @QueryParam(TIMEOUT_PARAM) Long timeout)
{
   log.debug("start> timeout = {}", timeout);
   Response resp = checkversionUUIDTS.run(uuid, version);
   if (resp != null) {
      return resp;
   }
   return Response.temporaryRedirect(URI.create("/" + uuid + "/" + rX + "/" +
      rY + "/" + rZ + "/" + version)).build();
}
```

JAX-RS: Structured data in API

```
//JSON data are sent in POST request
@POST
@Path("datasets/")
@Consumes(MediaType.APPLICATION_JSON)
public Response createEmptyDataset(DatasetDTO dataset) {
```

```
//JSON data are sent in GET response
@GET
@Path("datasets/")
@Produces(MediaType.APPLICATION_JSON)
public DatasetDTO createDataset(….
```

JAXB – Java Architecture for XML Bindings

- Used also for JSON

```
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class DatasetDTO {

    private String voxelType;
    private long[] dimensions;
    private int timepoints = 1;


....
```

CDI

- standard dependency injection framework included in Java EE 6 and higher.
- allows us to manage the lifecycle of stateful components via domain-specific lifecycle contexts and inject components (services) into client objects in a type-safe way.

CDI: Example

```
package org.acme.getting.started;

import javax.enterprise.context.ApplicationScoped;

@ApplicationScoped
public class GreetingService {

   public String greeting(String name) {
      return "hello " + name;
   }

}
```

```
package org.acme.getting.started;

import javax.inject.Inject;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

import org.jboss.resteasy.annotations.jaxrs.PathParam;

@Path("/hello")
public class GreetingResource {

   @Inject
   GreetingService service;

   @GET
   @Produces(MediaType.TEXT_PLAIN)
   @Path("/greeting/{name}")
   public String greeting(@PathParam String name) {
      return service.greeting(name);
   }

   @GET
   @Produces(MediaType.TEXT_PLAIN)
   public String hello() {
      return "hello";
   }
}
```

Quarkus

- https://quarkus.io
- MicroProfile – optimize J2EE to Microservices
  - JAX-RS, JAXB, CDI
- Supersonic Subatomic Java
- Full-stack Framework
- OpenJDK HotSpot, GraalVM
- Microservices
- Small footprint
- Reduced boot time

Quarkus: Simplified Hibernate ORM with Panache

- Makes mapping simple
- Active record.
- Repository.
- Advanced queries.
- Transactions.
- Lock management.
- Custom IDs
- Mocking

Panache: Active Record example.

```
@Entity
public class Person extends PanacheEntity {
   public String name;
   public LocalDate birth;
   public Status status;
}
```

```
// creating a person
Person person = new Person();
person.name = "Stef";
person.birth = LocalDate.of(1910, Month.FEBRUARY, 1);
person.status = Status.Alive;

// persist it
person.persist();

// note that once persisted, you don't need to explicitly save your entity: all
// modifications are automatically persisted on transaction commit.

// check if it's persistent
if(person.isPersistent()){
   // delete it
   person.delete();
}

// getting a list of all Person entities
List<Person> allPersons = Person.listAll();
```

```
// finding a specific person by ID
person = Person.findById(personId);

// finding a specific person by ID via an Optional
Optional<Person> optional = Person.findByIdOptional(personId);
person = optional.orElseThrow(() -> new NotFoundException());

// finding all living persons
List<Person> livingPersons = Person.list("status", Status.Alive);

// counting all persons
long countAll = Person.count();

// counting all living persons
long countAlive = Person.count("status", Status.Alive);

// delete all living persons
Person.delete("status", Status.Alive);

// delete all persons
Person.deleteAll();

// delete by id
boolean deleted = Person.deleteById(personId);

// set the name of all living persons to 'Mortal'
Person.update("name = 'Mortal' where status = ?1", Status.Alive);
```

Panache: Active Record example II

```
//All list methods have equivalent stream versions.
try (Stream<Person> persons = Person.streamAll()) {
   List<String> namesButEmmanuels = persons
      .map(p -> p.name.toLowerCase() )
      .filter( n -> ! "emmanuel".equals(n) )
      .collect(Collectors.toList());
}
```

```
@Entity
public class Person extends PanacheEntity {
   public String name;
   public LocalDate birth;
   public Status status;

   public static Person findByName(String name){
      return find("name", name).firstResult();
   }

   public static List<Person> findAlive(){
      return list("status", Status.Alive);
   }

   public static void deleteStefs(){
      delete("name", "Stef");
   }
}
```

Panache: Repository pattern

```
@ApplicationScoped
public class PersonRepository implements PanacheRepository<Person> {

  // put your custom logic here as instance methods

  public Person findByName(String name){
     return find("name", name).firstResult();
  }

  public List<Person> findAlive(){
     return list("status", Status.Alive);
  }

  public void deleteStefs(){
     delete("name", "Stef");
  }
}
```

```
@Inject
PersonRepository personRepository;

@GET
public long count(){
   return personRepository.count();
}
```

Quarkus: Getting Started

```
mvn io.quarkus:quarkus-maven-plugin:1.13.0.Final:create \
   -DprojectGroupId=vsb.java2.koz01 \
   -DprojectArtifactId=rest-getting-started \
   -DclassName="vsb.java2.rest.GreetingResource" \
   -Dpath="/hello"
```

```
@Path("/hello")
public class GreetingResource {

   @GET
   @Produces(MediaType.TEXT_PLAIN)
   public String hello() {
      return "hello";
   }
}
```

Quarkus: Package and run application

- ./mvnw package
  - rest-getting-started-1.0.0-SNAPSHOT.jar
  - quarkus-run.jar + quarkus-app/lib/
- java -jar target/quarkus-app/quarkus-run.jar

Quarkus: Native application

- With GrallVM installed
  - `./mvnw package -Pnative.`
- Linux executable with docker installed
  - `/mvnw package -Pnative -Dquarkus.native.container-build=true`
- Creating docker container
  - `./mvnw package -Pnative -Dquarkus.native.container-build=true -Dquarkus.container-image.build=true`

9th Lecture

- REST client
- GraalVM

REST client in Java: Libraries and Frameworks

- Apache CXF
- Jersey
- Spring RestTemplate
- Commons HTTP Client
- Apache HTTP Components (4.2) Fluent adapter
- OkHttp
- Ning Async-http-client
- Feign
- Retrofit
- Volley
- google-http
- Unirest
- Resteasy JakartaEE
- jcabi-http
- restlet
- rest-assured

https://stackoverflow.com/questions/221442/how-do-you-create-a-rest-client-for-java

Apache CXF

- JAX-RS 2.0 Client API
- Proxy-based API
- CXF WebClient API

pom.xml

```
<dependency>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-rt-rs-client</artifactId>
  <version>3.0.15</version>
</dependency>
```

Apache CFX

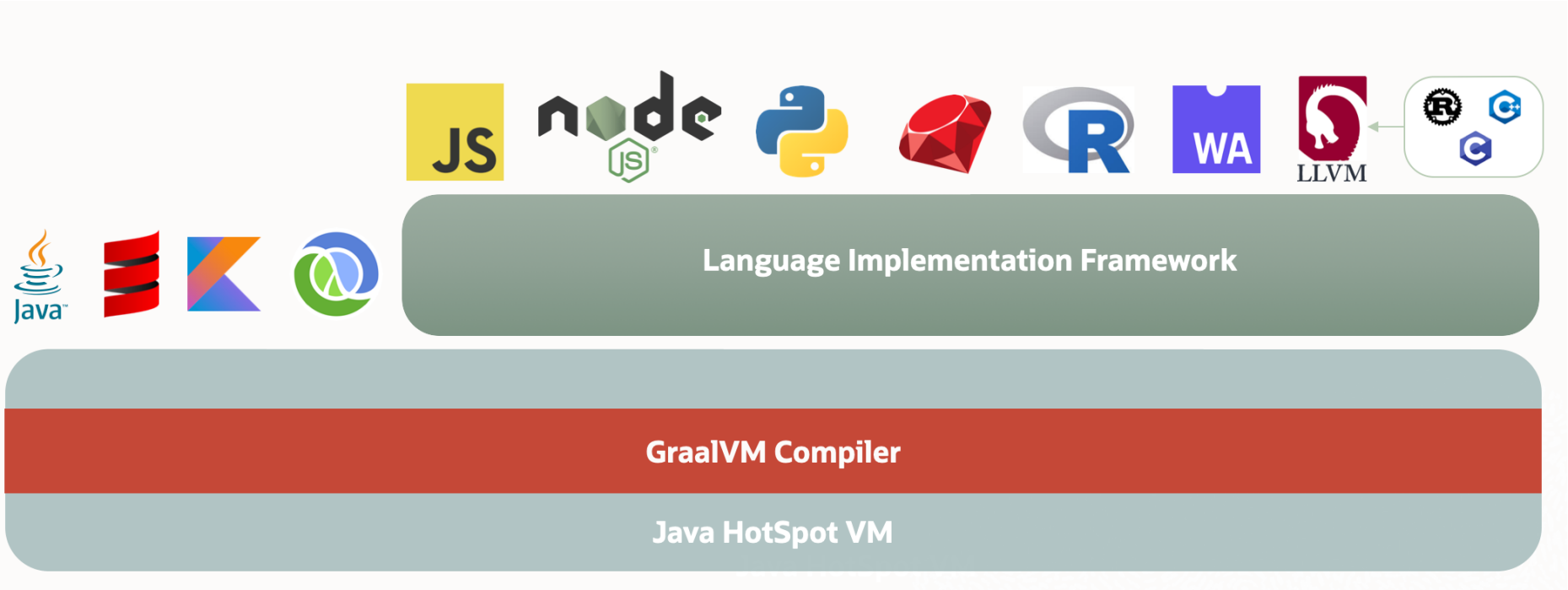
```
@Path("/bookstore")
public interface BookStore {
  @GET
  Books getAllBooks();

  @Path("{id}")
  BookResource getBookSubresource(@PathParam("id") long id) throws NoBookFoundException;
}
public interface BookResource {
  @GET
  Book getBook();
}
```

```
BookStore store = JAXRSClientFactory.create("http://bookstore.com", BookStore.class);
// (1) remote GET call to http://bookstore.com/bookstore
Books books = store.getAllBooks();
// (2) no remote call
BookResource subresource = store.getBookSubresource(1);
// {3} remote GET call to http://bookstore.com/bookstore/1
Book b = subresource.getBook();
```

GraalVM

- https://www.graalvm.org/
- A high-performance JDK distribution
- Also runtime for JavaScript, Ruby, Python, C
- Ahead –of-time compilation (AOT compilation)
- Base on OpenJDK 8 or 11

GraalVM: Architecture



GraalVM: Runtime Modes

- JVM Runtime Mode
- Native image
- Java on Truffle

GraalVM: Installation

- Download - https://github.com/graalvm/graalvm-ce-builds/releases
- add to PATH … <graalvm-directory>\bin or <graalvm-directory>/bin
- set JAVA_HOME … <graalvm-directory>

GraalVM: Run application

- Additional:
  - js
  - lli
  - gu
- Java … javac, java
- Java Script … js

```
js
> 1 + 2
3
```

GraalVM: Node.js

```
gu install nodejs
node -v
v14.16.1
```

```
npm install colors ansispan
```

```
const http = require("http");
const span = require("ansispan");
require("colors");

http.createServer(function (request, response) {
    response.writeHead(200, {"Content-Type": "text/html"});
    response.end(span("Hello Graal.js!".green));
}).listen(8000, function() { console.log("Graal.js server running at http://127.0.0.1:8000/".red); });

setTimeout(function() { console.log("DONE!"); process.exit(); }, 2000);
```

```
node app.js
```

GraalVM: LLVM

```
gu install llvm-toolchain
export LLVM_TOOLCHAIN=$(lli --print-toolchain-path)
```

```
#include <stdio.h>

int main() {
    printf("Hello from GraalVM!\n");
    return 0;
}
```

```
$LLVM_TOOLCHAIN/clang hello.c -o hello
./hello
```

GraalVM: Native Images

```
gu install native-image
```

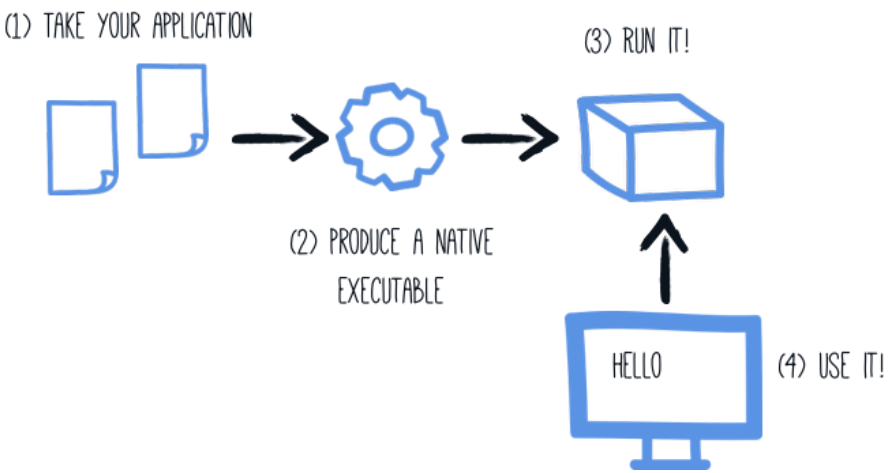
```
//HelloWorld.java
public class HelloWorld {
  public static void main() {
    System.out.println("Hello, World!");
  }
}
```

```
javac HellowWorld.java
native-image HelloWorld
```

```
./helloworld
Hello, World!
```

GraalVM: Build native image with Quarkus

- set GRAALVM_HOME in advance




```
./mnw package -Pnative
```

```
./target/<app>-runner
```

10th lecture

- Java security

Java Security

- https://docs.oracle.com/javase/9/security/toc.htm

Java Language Security a Bytecode Verification

- Type-safe,automatic memory management, garbage collection, and range-checking on arrays
- machine-independent bytecode representation
- A bytecode verifier
- Visibility modifiers
- Modules

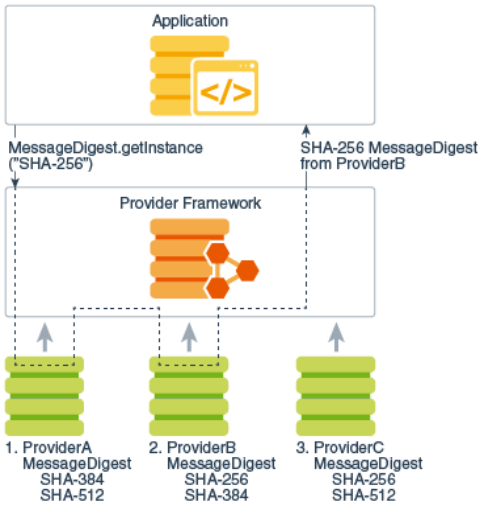
Basic Security Architecture

- API design principles
  - Implementation independence
  - Implementation interoperability
  - Algorithm extensibility

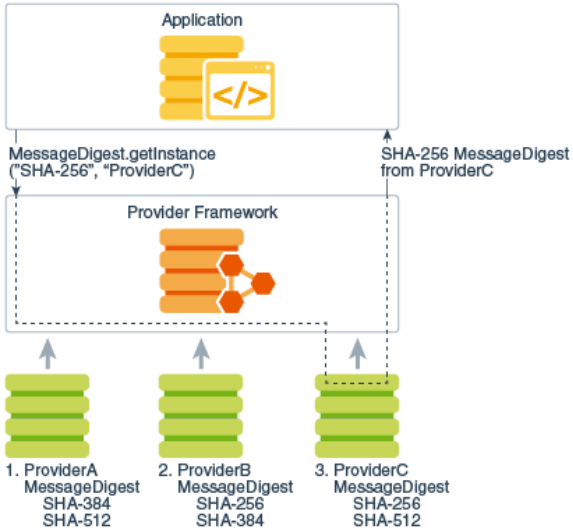
Basic Security Architecture: Security providers

- Java.security.Provider
  - getInstance

```
MessageDigest md =
    MessageDigest.getInstance("SHA-256");
```




```
MessageDigest md =
    MessageDigest.getInstance("SHA-256", "ProviderC");
```



Basic Security Architecture: File Locations

File Name or Tool Name	Location	**Description**
java.security	<java-home>/conf/security	Certain aspect for security
java.policy	<java-home>/conf/security	Default systwm policy file
Cryptographic policy directory	<java-home>/conf/security/policy	contains sets of jurisdiction policy files
cacerts	<java-home>/lib/security	a system-wide keystore with Certificate Authority (CA) and other trusted certificates
keytool, jarsigner, policytool Windows only: kinit, klist, ktab	<java-home>/bin	

Java Cryptography

- Message digest algorithms
- Digital signature algorithms
- Symmetric bulk and stream encryption
- Asymmetric encryption
- Password-based encryption (PBE)
- Elliptic Curve Cryptography (ECC)
- Key agreement algorithms
- Key generators
- Message Authentication Codes (MACs)
- Secure Random Number Generators

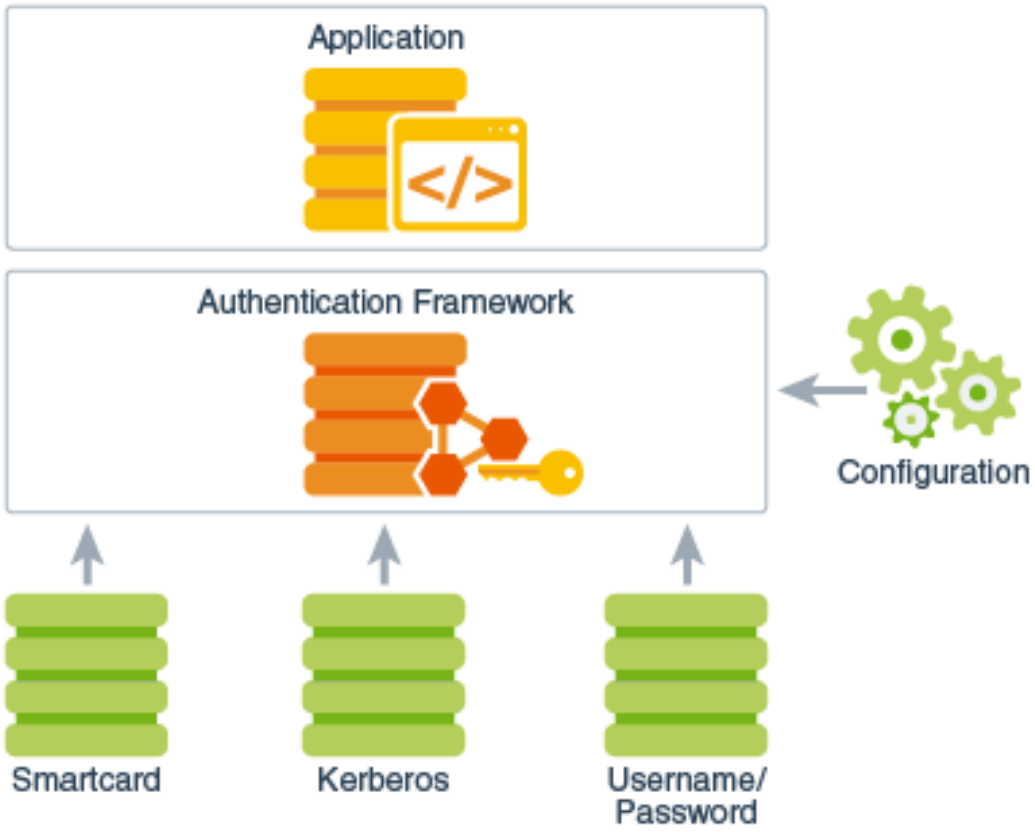
Java Cryptography: API packages

- java.security and java.security.*
- javax.crypto

Public Key Infrastructure

- Key and Certificate Storage
  - Java.security.KeyStore
  - Java.security.cert.CertStore
  - File cacerts
- Tools
  - Keytool
  - jarsigner

Authentication

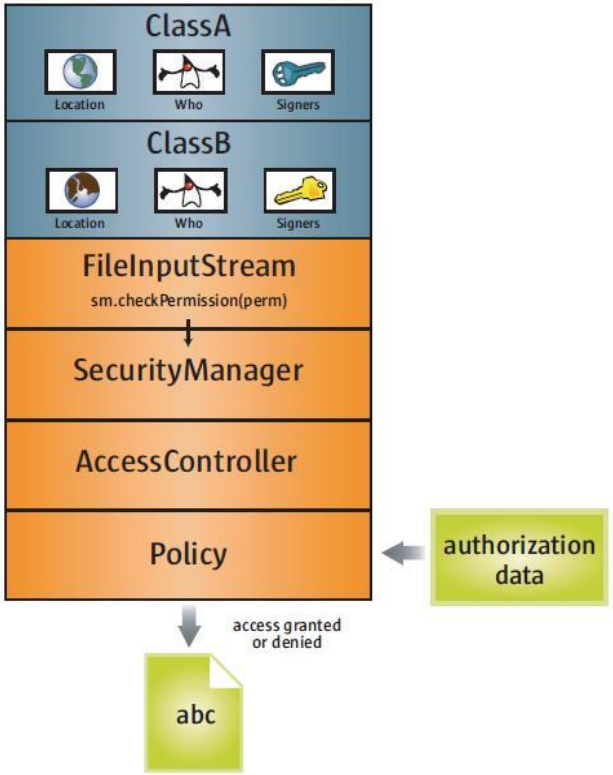


Secure Communication

- SSL, TLS, and DTLS Protocols
  - javax.net.ssl.SSLSocket, javax.net.ssl.SSLEngine
  - javax.net.ssl.KeyManager, TrustManager
  - Built-in providers: SSLv3, TLSv1, TLSv1.1, TLSv1.2, DTLSv1.0, DTLSv1.2
- Simple Authentication and Security Layer (SASL)

Access Control

- Permissions
  - java.security.Permission
- Security Policy
  - Java.security.Policy
- Access Control Enforcement




```
Permission perm = new java.io.FilePermission("/tmp/abc", "read");
SecurityManager sm = System.getSecurityManager();
if (sm != null) {
  sm.checkPermission(perm);
}
```

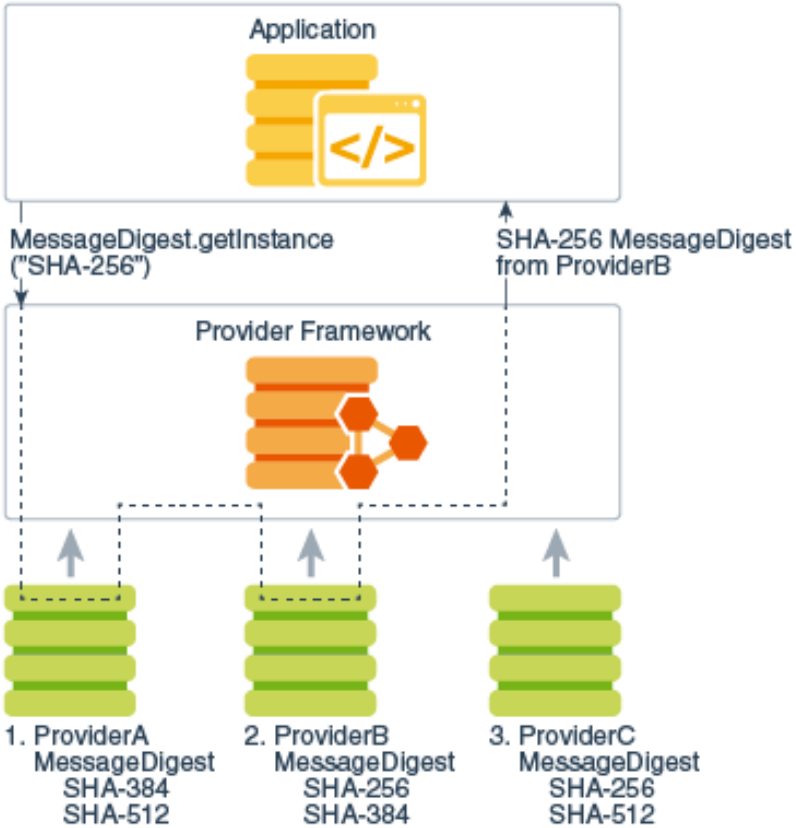
XML Signature

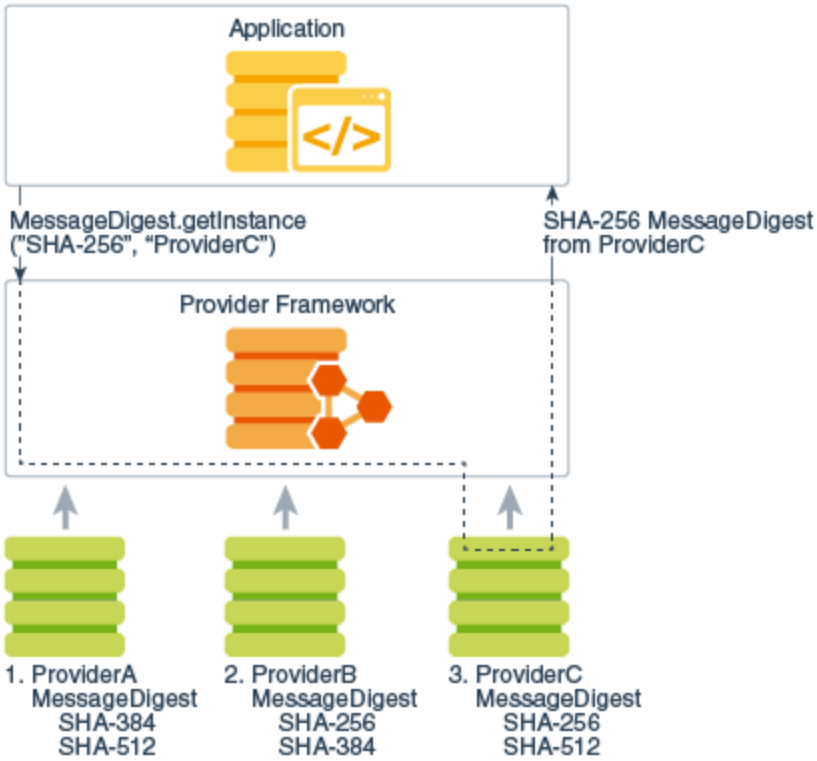
- java.xml.crypto
  - javax.xml.crypto
  - javax.xml.crypto.dsig
  - javax.xml.crypto.dsig.keyinfo
  - javax.xml.crypto.dsig.spec
  - javax.xml.crypto.dom
  - javax.xml.crypto.dsig.dom

Java Cryptography Architecture (JCA)

- Implementation independence and interoperability
- Algorithm independence and extensibility

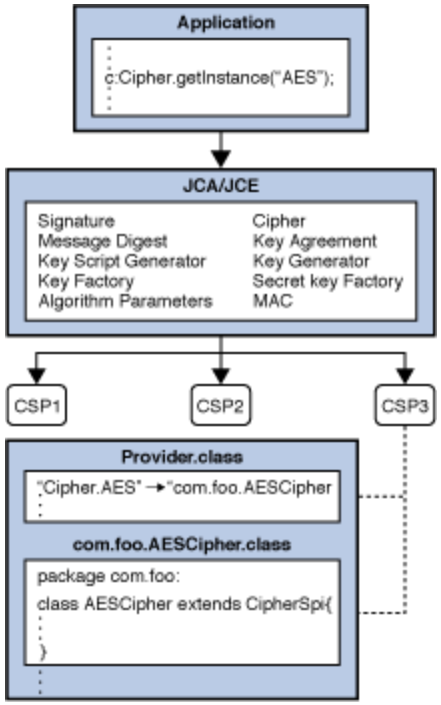
JCA: Provider Architecture





JCA: Getting an Instance of an Engine Class

```
import javax.crypto.*;

Cipher c = Cipher.getInstance("AES");
c.init(ENCRYPT_MODE, key);
```



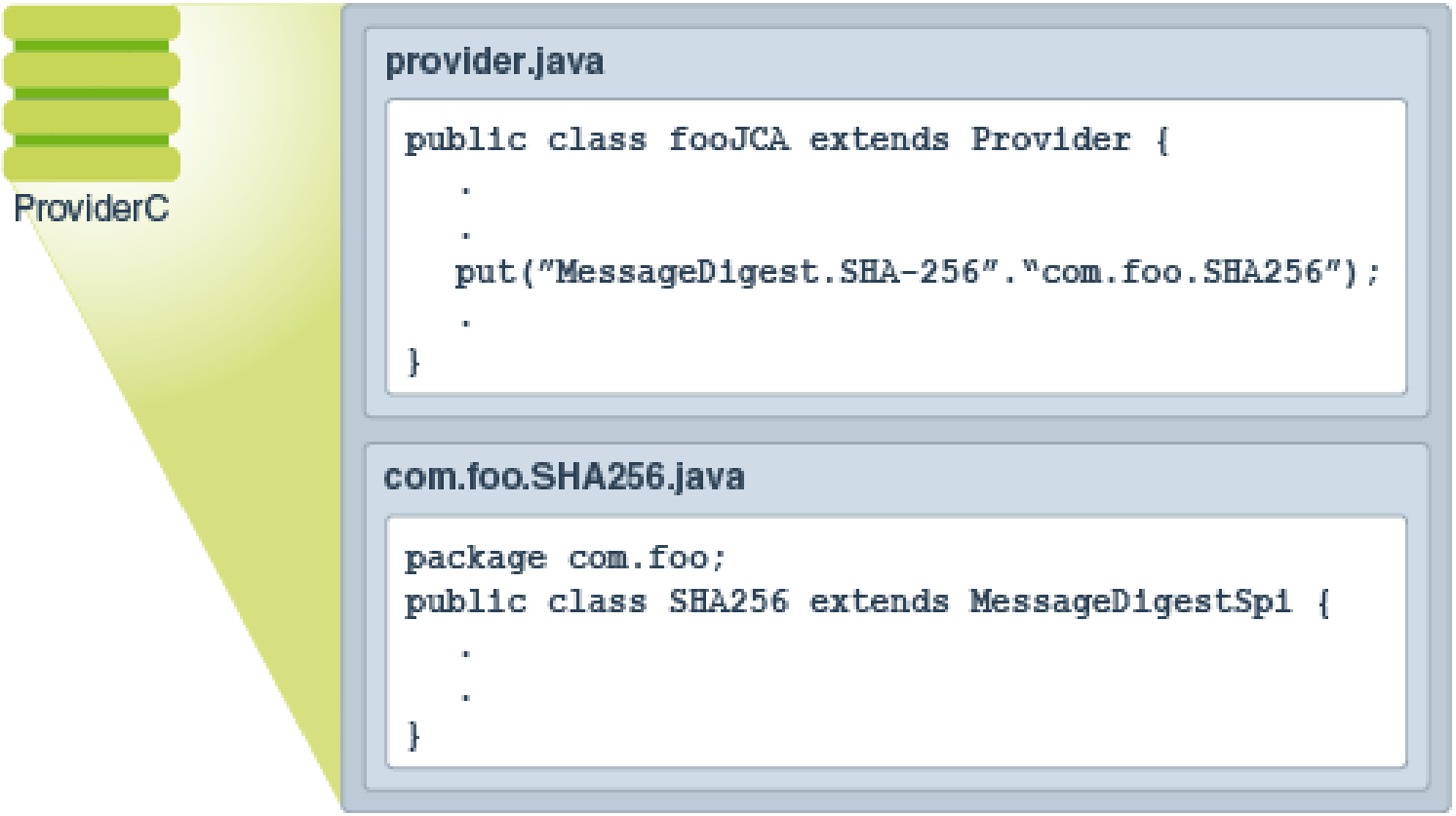
JCA: Engine Classes and Algorithms

- provides:
  - cryptographic operations (encryption, digital signatures, message digests, etc.),
  - generators or converters of cryptographic material (keys and algorithm parameters),
  - objects (keystores or certificates)

JCA: Engine Classes

- *SecureRandom*: used to generate random or pseudo-random numbers.
- *MessageDigest*: used to calculate the message digest (hash) of specified data.
- *Signature*: initialized with keys, these are used to sign data and verify digital signatures.
- *Cipher*: initialized with keys, these used for encrypting/decrypting data. There are various types of algorithms: symmetric bulk encryption (e.g. AES), asymmetric encryption (e.g. RSA), and password-based encryption (e.g. PBE).
- …

JCA: The Provider Class



JCA: Request Provider

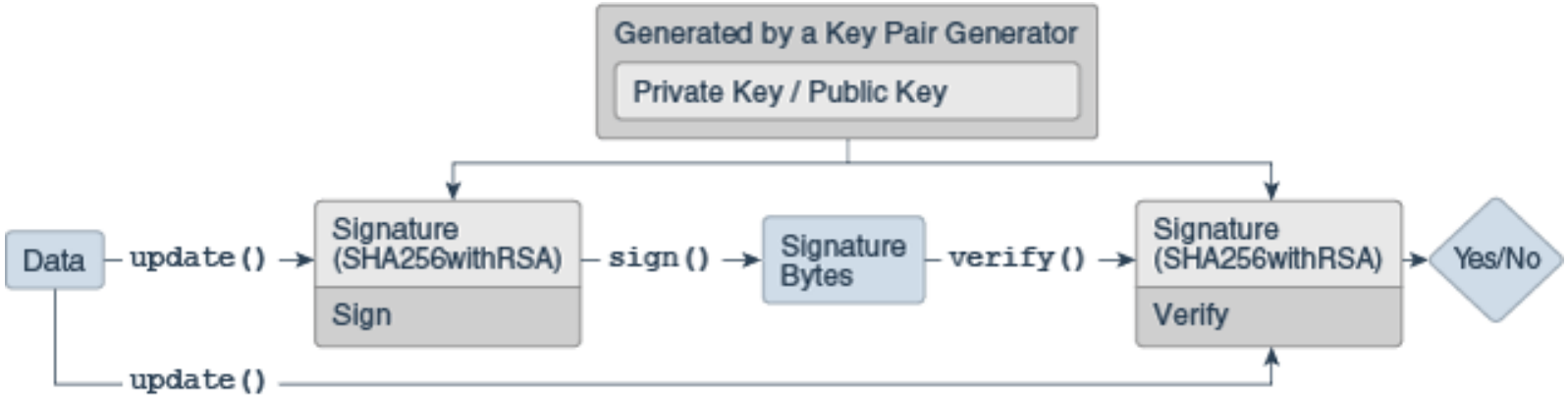
```
static EngineClassName getInstance(String algorithm)
   throws NoSuchAlgorithmException

static EngineClassName getInstance(String algorithm, String provider)
   throws NoSuchAlgorithmException, NoSuchProviderException

static EngineClassName getInstance(String algorithm, Provider provider)
      throws NoSuchAlgorithmException
```

```
MessageDigest md = MessageDigest.getInstance("SHA-256")
KeyAgreement ka = KeyAgreement.getInstance("DH", "SunJCE");
```

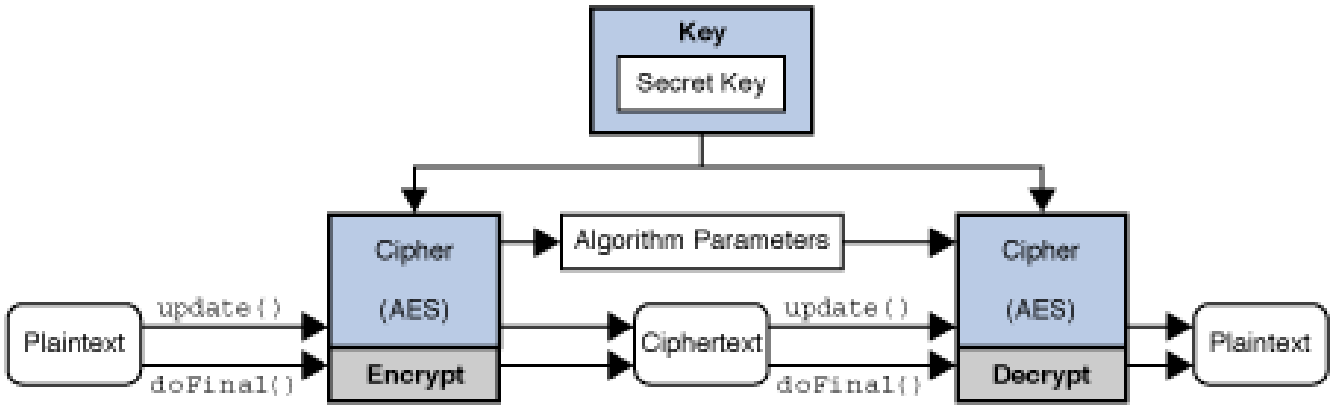
JCA: SecureRandom

JCA: MessageDigest

JCA: The Signature Class



JCA: The Cipher Class



JCA: The Cipher Class II

```
SecretKey myKey = ...
byte[] myAAD = ...
byte[] plainText = ...
int myTLen = ...
byte[] myIv = ...

GCMParameterSpec myParams = new GCMParameterSpec(myTLen, myIv);
Cipher c = Cipher.getInstance("AES/GCM/NoPadding");
c.init(Cipher.ENCRYPT_MODE, myKey, myParams);

// AAD is optional, if present, it must be supplied before any update/doFinal calls.
c.updateAAD(myAAD);  // if AAD is non-null
byte[] cipherText = new byte[c.getOutputSize(plainText.length)];
// conclusion of encryption operation
int actualOutputLen = c.doFinal(plainText, 0, plainText.length, cipherText);

// To decrypt, same AAD and GCM parameters must be supplied
c.init(Cipher.DECRYPT_MODE, myKey, myParams);
c.updateAAD(myAAD);
byte[] recoveredText = c.doFinal(cipherText, 0, actualOutputLen);

// MUST CHANGE IV VALUE if the same key were to be used again for encryption
byte[] newIv = ...;
myParams = new GCMParameterSpec(myTLen, newIv);
```

JCA: CipherInputStream

```
try (FileInputStream fis = new FileInputStream("/tmp/a.txt");
CipherInputStream cis = new CipherInputStream(fis, cipher1);
FileOutputStream fos = new FileOutputStream("/tmp/b.txt")) {
   byte[] b = new byte[8];
   int i = cis.read(b);
   while (i != -1) {
      fos.write(b, 0, i);
      i = cis.read(b);
   }
}
```

JCA: CipherOutputStream

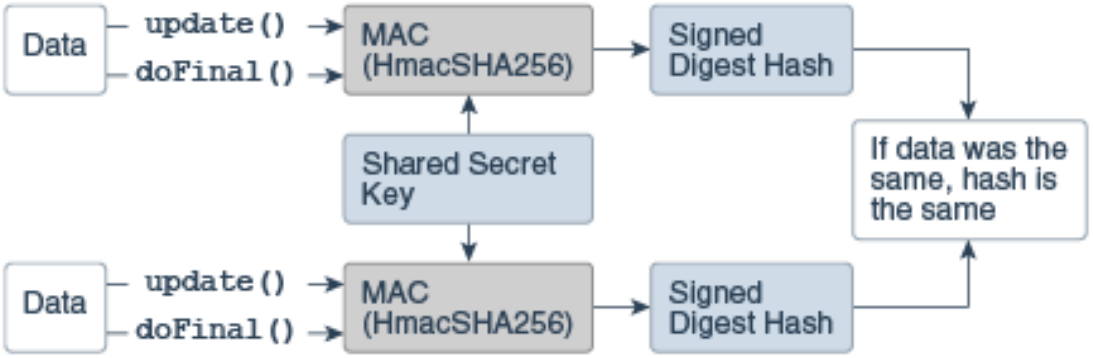
```
try (FileInputStream fis = new FileInputStream("/tmp/a.txt");
      FileOutputStream fos = new FileOutputStream("/tmp/b.txt");
      CipherOutputStream cos = new CipherOutputStream(fos, cipher1)) {
   byte[] b = new byte[8];
   int i = fis.read(b);
   while (i != -1) {
      cos.write(b, 0, i);
      i = fis.read(b);
   }
   cos.flush();
}
```

JCA: SealedObject class

```
// create Cipher object
   // NOTE: sKey is assumed to refer to an already-generated
   // secret AES key.
   Cipher c = Cipher.getInstance("AES");
   c.init(Cipher.ENCRYPT_MODE, sKey);


   // do the sealing
SealedObject so = new SealedObject("This is a secret", c);
```

```
c.init(Cipher.DECRYPT_MODE, sKey);
   try {
      String s = (String)so.getObject(c);
   } catch (Exception e) {
      // do something
        };
```

JCA: The Mac Class



JCA: Key Interface

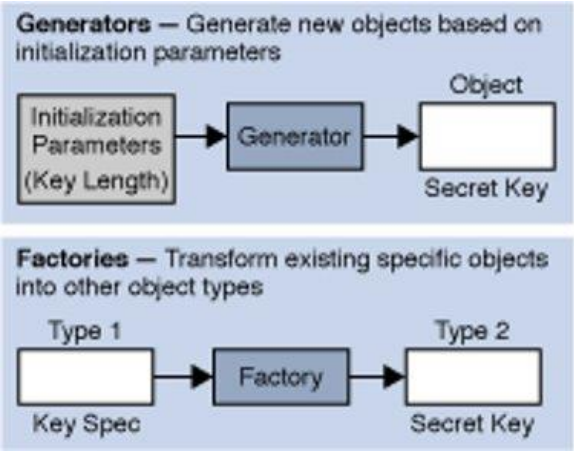
- String getAlgorithm()
- byte[] getEncoded()
- String getFormat()
- SecretKey, PrivateKey, PublicKey
- KeyPair:
  - PrivateKey getPrivate()
  - PublicKey getPublic()

JCA: Key Specification Interfaces and Classes

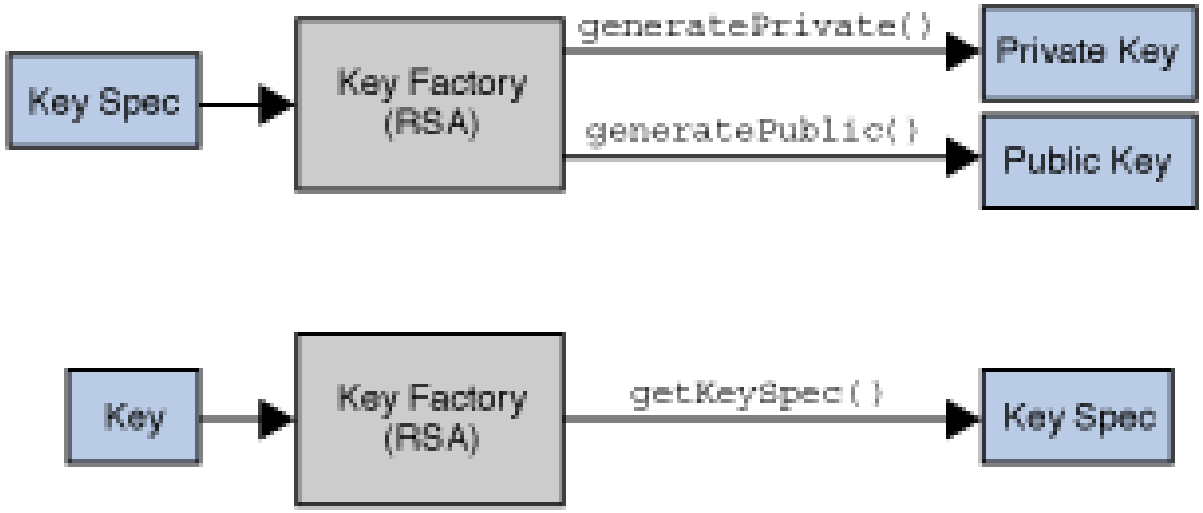
- Portable form – base type KeySpec interface
- KeyFactory and SecretKeyFactory – specification to key and back
- SecretKetSpec, EncodedKeySpec, DESKeySpec, ….

JCA: Generators and Factories

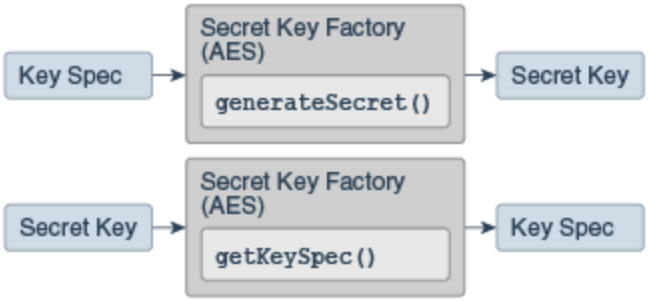
- Engine class – method getInstance



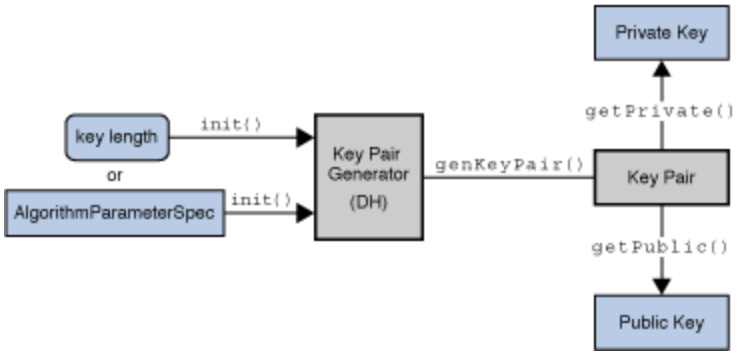
JCA: KeyFactory



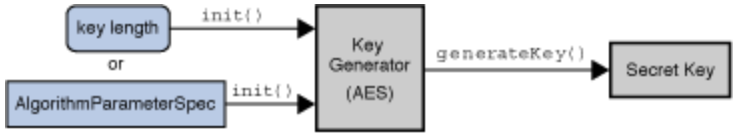
JCA: SecretKeyFactory



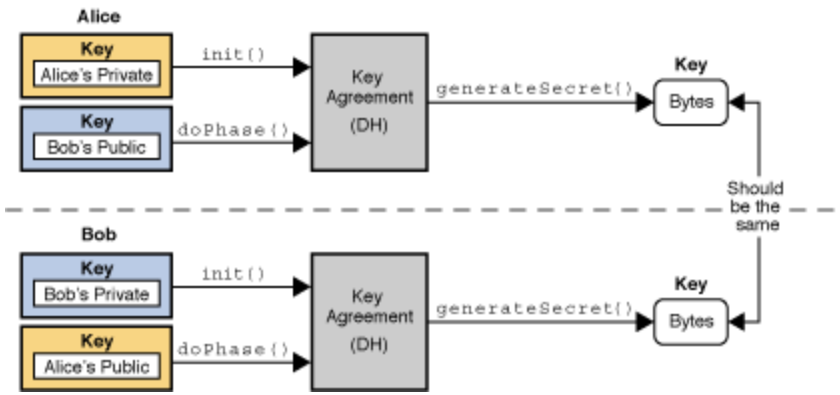
JCA: KeyPairGenerator



JCA: KeyGenerator

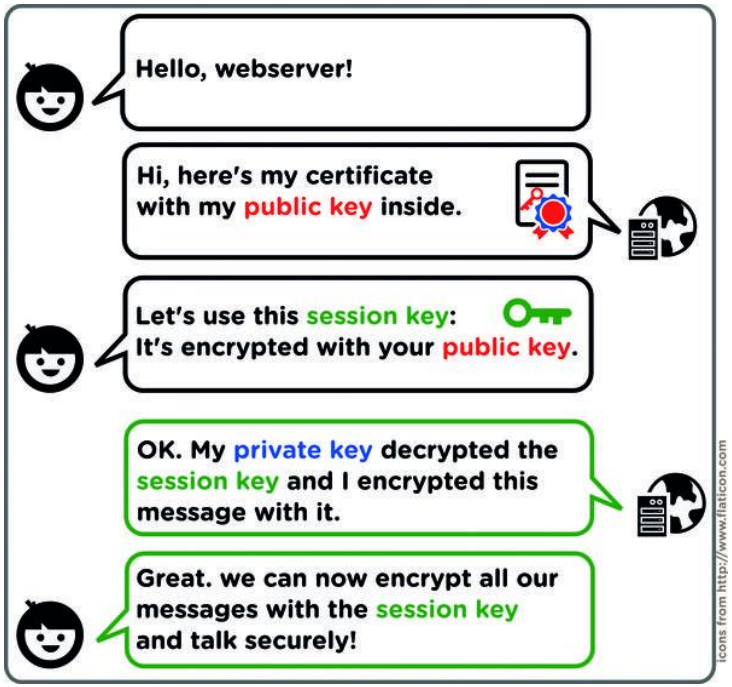


JCA: KeyAgreement



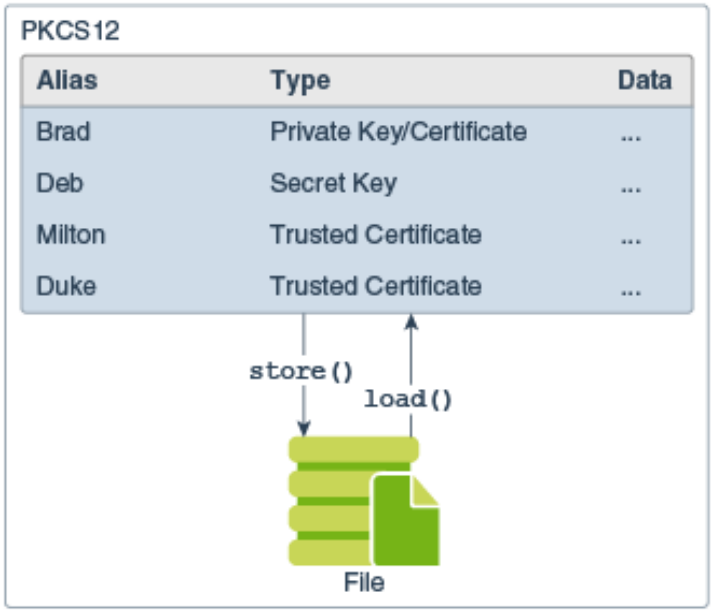
JCA: Key Management

- Database named "keystore" can be used to manage a repository of **keys** and **certificates -**
  - User - <user.home>/.keystore
  - System - <java-home>/lib/security/cacerts
- keytool, jarsigner
- A **certificate** is a digitally signed statement from one entity, saying that the public key of some other entity has a particular value.



JCA: KeyStore Class

- supplies well-defined interfaces to access and modify the information in a keystore.



JCA: Working with key store

- getInstance
- Load
  - final void load(InputStream stream, char[] password)
  - final void load(KeyStore.LoadStoreParameter param)
- final Enumeration aliases() - Enumeration.toIteratorfinal
- boolean isKeyEntry(String alias), final boolean isCertificateEntry(String alias)

JCA: Working with key store

```
final void setCertificateEntry(String alias, Certificate cert)

final void setKeyEntry(String alias,
                       Key key,
                       char[] password,
                       Certificate[] chain)

final void setKeyEntry(String alias,
                       byte[] key,
                       Certificate[] chain)

final void deleteEntry(String alias) ....
```

JCA: Working with key store II

```
final Key getKey(String alias, char[] password)

final Certificate getCertificate(String alias)


final void store(OutputStream stream, char[] password)
```

JCA: Algortihm Parameter Classes

- *AlgorithmParameters* vs *AlgorithmParameterSpec*

JCA: AlgorithmParameters

- getInstance

```
void init(AlgorithmParameterSpec paramSpec)
void init(byte[] params)
void init(byte[] params, String format)

byte[] getEncoded()

AlgorithmParameterSpec getParameterSpec(Class paramSpec)
```

JCA: AlgorithmParameterGenerator

- getInstance

```
//algorithm independent
void init(int size, SecureRandom random);
void init(int size)

//algorithm-specific
void init(AlgorithmParameterSpec genParamSpec,
                SecureRandom random)
void init(AlgorithmParameterSpec genParamSpec)


AlgorithmParameters generateParameters()
```

JCA: CertificateFactory Class

- defines the functionality of a certificate factory, which is used to generate certificate and certificate revocation list (CRL) objects from their encoding.
  - final Certificate generateCertificate(InputStream inStream)
  - final CRL generateCRL(InputStream inStream)

Thank you for your attention

Ing. Jan Kožusznik, Ph.D.

+420 724 474 535

jan@kozusznik.cz

www.kozusznik.cz