## Effective Java 3-dagars Hands-On kurs

**Bli en seriös Java-programmerare på 3 dagar! Som deltagare i denna kurs kommer du att få lära dig hur man skriver robust och väldesignad Java-kod. Du får lära dig att implementera ett antal designmönster och bästa praxis för att skapa hög-kvalitativ kod som är effektiv i termer av prestanda, underhåll och läsbarhet.**

**Kursen följer de riktlinjer som formats av Joshua Block i boken ”Effective Java”, och täcker systematiskt in ett antal viktiga områden inom modern systemutveckling med Java. Kursen är uppdaterad med de nya möjligheter som Java 8 erbjuder, såsom funktionsorienterad programmering med lambda uttryck, utökad syntax för interface samt genomgång av det nya biblioteket Stream.**

**Exempel och övningar presenteras med UML-diagram och korrekt robust Java-kod, kompletterade med ett stort antal fördefinierade enhetstester.**

**Målgrupp**

Java-programmerare som vill fördjupa sina kunskaper inom Java-programmering med målet att kunna producera robust kod av hög kvalitét.

Programmerare med erfarenhet från en tidigare Java-version, som behöver fräscha upp sina kunskaper och få en djupare förståelse av programspråket och de viktigaste Java-biblioteken.

Erfarna C++ / C# programmerare som vill ha en introduktion till Java i full fart!

**Hands-On**

Till varje teori-avsnitt finns motsvarande kompletta exempel i Java-kod färdig att köras. Dessutom innehåller kursen ett antal övningar som arbetas igenom och diskuteras under kursens gång. All Java-kod, exempel och lösningsförslag, ingår i kursdokumentationen som varje deltagare har tillgång till.

## Dokumentation

Kopior på PPT-bilder

“Effective Java 2nd ed” Joshua Bloch

Komplett kod till alla exempel/övningar

## Contents:

**1. Object Orientation using UML**

* UML
* Classes and Objects
* UML implemented in Java

**2. Unit Testing using JUnit4/TestNG**

* JUnit4/TestNG architecture
* Annotations for tests
  + @Test
  + @Before
  + @After
* Integration of unit test framework in Eclipse/NetBeans
* Running separate tests
* Running data driven tests
* Running test suites

**3. Overriding Object Methods**

* equals
* hashCode
* toString
* Implementation of interface Comparable<T>

**4. Software Design - Classes and Interfaces**

* Level of access
  + Encapsulation
  + Public methods and private attributes
  + Immutable classes – value classes
* Class design
  + Inheritance vs. composition
  + Interfaces supporting default and static methods
  + Abstract base classes vs. interfaces
* Creating and destroying objects
  + Static factory methods vs. constructors
  + Design Pattern Singleton
  + Reduce need for garbage collection
* Nested classes
  + Static classes
  + Inner classes
  + Anonymous classes
  + Local classes

**5. Formatting Text**

* java.util.Formatter
* Using method format for String objects
* Parameter handling in format strings

**6. General Programming**

* Avoid floating point types for exact calculations
* Performance for wrapper classes vs. primitive types
* Performance issues for String
* Validating method arguments
* Defensive copying of return values
* Avoid returning null

**7. Generics and Typesafe Collections**

* Generics
  + Parameterized types
  + Arrays vs. Generics
* Type safety in collection classes
  + Erasure guarantees backwards compatibility

**8. The Collections Framework in Detail**

* Basic interfaces
  + List, Set, SortedSet, Queue, Map, SortedMap
* Basic implementation classes
  + ArrayList, HashSet, TreeSet, HashMap, TreeMap
* Iterator vs. for:each-loop
* Specialized interfaces
  + Deque, NavigableSet, NavigableMap
* Implementation classes for specialized interfaces
  + ArrayDeque, PriorityQueue
* Performance aspects for implementation classes
* Synchronized classes

**9. Enums**

* Enum defining constants
* Enum defining attributes and methods
* Collection classes using Enum based keys
  + EnumSet,EnumMap

**10. Annotations**

* + Annotation as a Java type
  + Rules for usage
  + Retention policies
  + Annotations vs. naming conventions

**11. Exception handling**

* Overview exceptions
  + Runtime exceptions
  + Checked exceptions
  + Chained exceptions
* Exception or special return value
* Choice of exception type
* Use standard exceptions
* Support logging info in exception message

**12. Serialization**

* Serialization mechanism in Java
* Customization of serialization
* Externalizable for special purposes
* Version handling for Serializable classes

**13. Lambda Expressions**

* Functional programming using lambda expressions
  + Syntax for Lambda Expressions
  + Deferred execution
  + Capture of variables
* Functional Interfaces
  + **Built in support in Java APIs**
* **Predefined Interfaces supporting Lambda Expressions**
  + **Function, Operator, Supplier, Consumer, Predicate**
* Method references

**14. The Streams API**

* Stream creation from existing sources
* Iterations using stream operations
* Streams using lambda expressions
* Composition and reduction of streams
* Bulk operations for collections
* Enabling parallel processing of streams