Java Certification

Source - https://education.oracle.com/education/pdf/JavaCertificationMap.pdf

The Oracle Certification track now consists of levels – Associate (OCA)

- Professional (OCP)
- Master (OCM)
- & a few others - Talk about complexity

Java 8

- Oracle Certified Associate, Java SE 8 Programmer 1Z0-808
- Oracle Certified Professional, Java SE 8 Programmer II, 1Z0-809
- You have other upgrade options from earlier versions
- There is NO Oracle Certified Master for Java SE 8
    - The last master certification for Java - Oracle Certified Master, Java SE 6 Developer 1Z0-855 (includes an assignment and an essay) - is discontinued - The Oracle Certified Master, Java SE 6 Developer certification retires on November 30, 2016. - At this time, there is no plan to release a new version of the certification. Through careful job task analysis with the Java SE 8 certification release, it was determined that the OCM Developer role was no longer appropriate to keep as a separate certification and the that skills of that role should be included in the Programmer certifications.

Upgrade to Java SE 8 OCP (Java SE 6 and all prior versions)

From https://education.oracle.com/pls/web_prod-plq-dad/db_pages.getpage?page_id=5001&get_params=p_exam_id:1Z0-813

TOPICS

Language Enhancements

- Develop code that uses String objects in the switch statement, binary literals, and numeric literals, including underscores in literals
- Develop code that uses try-with-resources statements, including using classes that implement the AutoCloseable interface
- Develop code that handles multiple Exception types in a single catch block
- Use static and default methods of an interface including inheritance rules for a default method

Concurrency

- Use classes from the java.util.concurrent package including CyclicBarrier and CopyOnWriteArrayList with a focus on the advantages over and differences from the traditional java.util collections
- Use Lock, ReadWriteLock, and ReentrantLock classes in the java.util.concurrent.locks and java.util.concurrent.atomic packages to support lock-free thread-safe programming on single variables
- Use Executor, ExecutorService, Executors, Callable, and Future to execute tasks using thread pools
- Use the parallel Fork/Join Framework

Localization

- Describe the advantages of localizing an application and developing code that defines, reads, and sets the locale with a Locale object
- Build a resource bundle for a locale and call a resource bundle from an application
- Create and manage date- and time-based events by using LocalDate, LocalTime, LocalDateTime, Instant, Period, and Duration, including a combination of date and time in a single object
- Format dates, numbers, and currency values for localization with the NumberFormat and DateFormat classes, including number and date format patterns
- Work with dates and times across time zones and manage changes resulting from daylight savings

Java File I/O (NIO.2)

- Operate on file and directory paths by using the Paths class Check, delete, copy, or move a file or directory by using the Files class Recursively access a directory tree by using the DirectoryStream and FileVisitor interfaces
- Find a file by using the PathMatcher interface, and use Java SE 8 I/O improvements, including Files.find(), Files.walk(), and lines() methods Observe the changes in a directory by using the WatchService interface

Lambda

- Define and write functional interfaces and describe the interfaces of the java.util.function package
- Describe a lambda expression; refactor the code that uses an anonymous inner class to use a lambda expression; describe type inference and target typing
- Develop code that uses the built-in interfaces included in the java.util.function package, such as Function, Consumer, Supplier, UnaryOperator, Predicate, and Optional APIs, including the primitive and binary variations of the interfaces

- Develop code that uses a method reference, including refactoring a lambda expression to a method reference

Java Collections

- Develop code that uses diamond with generic declarations
- Develop code that iterates a collection, filters a collection, and sorts a collection by using lambda expressions
- Search for data by using methods, such as findFirst(), findAny(), anyMatch(), allMatch(), and noneMatch()
- Perform calculations on Java Streams by using count, max, min, average, and sum methods and save results to a collection by using the collect method and Collector class, including the averagingDouble, groupingBy, joining, partitioningBy methods
- Develop code that uses Java SE 8 collection improvements, including the Collection.removeIf(), List.replaceAll(), Map.computeIfAbsent(), and Map.computeIfPresent() methods
- Develop code that uses the merge(), flatMap(), and map() methods on Java Streams

Java Streams

- Describe the Stream interface and pipelines; create a stream by using the Arrays.stream() and IntStream.range() methods; identify the lambda operations that are lazy
- Develop code that uses parallel streams, including decomposition operation and reduction operation in streams

Java New Features Course Objectives

Introducing Lambda Expressions

- Describing the purpose of an anonymous inner class
- Describing drawbacks to anonymous inner classes
- Describing the components of a lambda expression
- Defining a functional interface
- Creating programs that use lambda expressions

A Case for Lambda Expressions

- Discussing the reasons for adding lambda expressions to the Java language
- Reviewing the standard way of extracting data in Java
- Refactoring code to reduce redundancy
- Refactoring code to use inner classes
- Refactoring code to use lambda expressions
- Listing the benefits of lambda expressions

Filtering Collections with Lambdas

- Iterating though a collection with forEach
- Iterating through a collection using lambda syntax
- Describing the Stream interface
- Filtering a collection using lambda expressions
- Calling an existing method using a method reference
- Chaining multiple methods together
- Comparing function and imperative programming
- Defining pipelines in terms of lambdas and collections

Using Built in Lambda Types

- Listing the built in interfaces included in java.util.function
- Determining true or false with a Predicate
- Processing an object and return nothing with Consumer
- Processing one object and return another with Function
- Generating a new object with Supplier
- Using primitive versions of the base interfaces
- Using binary versions of the base interfaces

Collection Operations with Lambda

- Extracting data from an object using map
- Searching for data using search methods
- Describing the types of stream operations
- Describing the Optional class
- Performing calculations using methods
- Describing lazy processing
- Sorting a stream
- Saving results to a collection using the collect method
- Parallel Streams
- Reviewing the key characteristics of streams

Contrasting old style loop operations with streams

- Describing how to make a stream pipeline execute in parallel
- Listing the key assumptions needed to use a parallel pipeline
- Defining reduction
- Describing why reduction requires an associative function
- Calculating a value using reduce
- Describing the process for decomposing and then merging work

Lambda Cookbook

- Modifying a list using removeIf
- Updating a list using replaceAll
- Updating a map using computeIfAbsent, computerIfPresent, and merge

- Sending the keys and values from a map to a stream
- Reading a file to a stream
- Reading a text file into an ArrayList
- List, walk, and search a directory structure using a stream
- Flattening a stream using flatMap

Method Enhancements

- Considering the importance of building good libraries
- Using static methods in Interfaces
- Using default methods
- Understanding default method inheritance rules

Using the Date/Time API: Working with Local Dates and Times

- Listing the goals of the Date/Time API (JSR-310)
- Creating and manage date-based events
- Creating and manage time-based events
- Combining date and time into a single object

Using the Date/Time API: Working with Time Zones

- Working with dates and times across time-zones and manage changes resulting from daylight savings

Using the Date/Time API: Working with Date and Time Amounts

- Defining and create timestamps, periods and durations
- Applying formatting to local and zoned dates and times

JavaScript on Java with Nashorn: Creating and executing shell scripts

- Creating and execute shell scripts using JavaScript and Nashorn
- JavaScript on Java with Nashorn: Writing JavaScript Applications
- Developing JavaScript applications that leverage Java code using Nashorn
- JavaScript on Java with Nashorn: Writing JavaFX Applications Using

JavaScript

- Running JavaScript script from Java applications usingJSR-223
- Prototype JavaFX applications using Nashorn and JavaScript

Intro to Mission Control

- Describing JMX and Managed Beans with Mission Control
- Monitoring CPU utilization with Mission Control
- Analyzing JVM characteristics with Mission Control
- Analyzing heap memory with Mission Control
- Intro to Flight Recorder

Describing the Java Flight Recorder

- Describing the Java Flight Recorder Architecture
- Starting a Java Flight Recording
- Managing a Java Flight Recording
- Analyzing a Java Flight Recording

Java SE 6 Programmer Certified Professional

Section 1: Declarations, Initialization and Scoping

- Develop code that declares classes (including abstract and all forms of nested classes), interfaces, and enums, and includes the appropriate use of package and import statements (including static imports).
- Develop code that declares an interface. Develop code that implements or extends one or more interfaces.
- Develop code that declares an abstract class. Develop code that extends an abstract class.
- Develop code that declares, initializes, and uses primitives, arrays, enums, and objects as static, instance, and local variables. Also, use legal identifiers for variable names.
- Given a code example, determine if a method is correctly overriding or overloading another method, and identify legal return values (including covariant returns), for the method.
- Given a set of classes and superclasses, develop constructors for one or more of the classes. Given a class declaration, determine if a default constructor will be created, and if so, determine the behavior of that constructor. Given a nested or non-nested class listing, write code to instantiate the class.

Section 2: Flow Control

- Develop code that implements an if or switch statement; and identify legal argument types for these statements.
- Develop code that implements all forms of loops and iterators, including the use of for, the enhanced for loop (for-each), do, while, labels, break, and continue; and explain the values taken by loop counter variables during and after loop execution.
- Develop code that makes use of assertions, and distinguish appropriate from inappropriate uses of assertions.
- Develop code that makes use of exceptions and exception handling clauses (try, catch, finally), and declares methods and overriding methods that throw exceptions.
- Recognize the effect of an exception arising at a specified point in a code fragment. Note that the exception may be a runtime exception, a checked exception, or an error.
- Recognize situations that will result in any of the following being thrown: ArrayIndexOutOfBoundsException,ClassCastException,

IllegalArgumentException, IllegalStateException, NullPointerException, NumberFormatException, AssertionError, ExceptionInInitializerError, StackOverflowError or NoClassDefFoundError. Understand which of these are thrown by the virtual machine and recognize situations in which others should be thrown programatically.

Section 3: API Contents

- Develop code that uses the primitive wrapper classes (such as Boolean, Character, Double, Integer, etc.), and/or autoboxing & unboxing. Discuss the differences between the String, StringBuilder, and StringBuffer classes.
- Given a scenario involving navigating file systems, reading from files, writing to files, or interacting with the user, develop the correct solution using the following classes (sometimes in combination), from java.io: BufferedReader, BufferedWriter, File, FileReader, FileWriter, PrintWriter, and Console.
- Use standard J2SE APIs in the java.text package to correctly format or parse dates, numbers, and currency values for a specific locale; and, given a scenario, determine the appropriate methods to use if you want to use the default locale or a specific locale. Describe the purpose and use of the java.util.Locale class.
- Write code that uses standard J2SE APIs in the java.util and java.util.regex packages to format or parse strings or streams. For strings, write code that uses the Pattern and Matcher classes and the String.split method. Recognize and use regular expression patterns for matching (limited to: . (dot), * (star), + (plus), ?, \d, \s, \w, [], ()). The use of *, +, and ? will be limited to greedy quantifiers, and the parenthesis operator will only be used as a grouping mechanism, not for capturing content during matching. For streams, write code using the Formatter and Scanner classes and the PrintWriter.format/printf methods. Recognize and use formatting parameters (limited to: %b, %c, %d, %f, %s) in format strings.

Section 4: Concurrency

- Write code to define, instantiate, and start new threads using both java.lang.Thread and java.lang.Runnable.
- Recognize the states in which a thread can exist, and identify ways in which a thread can transition from one state to another.
- Given a scenario, write code that makes appropriate use of object locking to protect static or instance variables from concurrent access problems.

Section 5: OO Concepts

- Develop code that implements tight encapsulation, loose coupling, and high cohesion in classes, and describe the benefits.
- Given a scenario, develop code that demonstrates the use of polymorphism. Further, determine when casting will be necessary and recognize compiler vs. runtime errors related to object reference casting.
- Explain the effect of modifiers on inheritance with respect to constructors, instance or static variables, and instance or static methods.

- Given a scenario, develop code that declares and/or invokes overridden or overloaded methods and code that declares and/or invokes superclass, or overloaded constructors.
- Develop code that implements "is-a" and/or "has-a" relationships.

Section 6: Collections / Generics

- Given a design scenario, determine which collection classes and/or interfaces should be used to properly implement that design, including the use of the Comparable interface.
- Distinguish between correct and incorrect overrides of corresponding hashCode and equals methods, and explain the difference between == and the equals method.
- Write code that uses the generic versions of the Collections API, in particular, the Set, List, and Map interfaces and implementation classes. Recognize the limitations of the non-generic Collections API and how to refactor code to use the generic versions. Write code that uses the NavigableSet and NavigableMap interfaces.
- Develop code that makes proper use of type parameters in class/interface declarations, instance variables, method arguments, and return types; and write generic methods or methods that make use of wildcard types and understand the similarities and differences between these two approaches.
- Use capabilities in the java.util package to write code to manipulate a list by sorting, performing a binary search, or converting the list to an array. Use capabilities in the java.util package to write code to manipulate an array by sorting, performing a binary search, or converting the array to a list. Use the java.util.Comparator and java.lang.Comparable interfaces to affect the sorting of lists and arrays. Furthermore, recognize the effect of the "natural ordering" of primitive wrapper classes and java.lang.String on sorting.

Section 7: Fundamentals

- Given a code example and a scenario, write code that uses the appropriate access modifiers, package declarations, and import statements to interact with (through access or inheritance) the code in the example.
- Given an example of a class and a command-line, determine the expected runtime behavior.
- Determine the effect upon object references and primitive values when they are passed into methods that perform assignments or other modifying operations on the parameters.
- Given a code example, recognize the point at which an object becomes eligible for garbage collection, determine what is and is not guaranteed by the garbage collection system, and recognize the behaviors of the Object.finalize() method.
- Given the fully-qualified name of a class that is deployed inside and/or outside a JAR file, construct the appropriate directory structure for that class. Given a

code example and a classpath, determine whether the classpath will allow the code to compile successfully.
- Write code that correctly applies the appropriate operators including assignment operators (limited to: =, +=, -=), arithmetic operators (limited to: +, -, *, /, %, ++, --), relational operators (limited to: <, <=, >, >=, ==, !=), the instanceof operator, logical operators (limited to: &, |, ^, !, &&, ||), and the conditional operator (? :), to produce a desired result. Write code that determines the equality of two objects or two primitives.

Preparation Course For Java Certification 1

Getting Started

- Examine Java technology
- Analyze a simple Java technology application
- Execute a Java technology application

Object-Oriented Programming

- Define modeling concepts: abstraction, encapsulation, and packages
- Discuss Java technology application code reuse
- Define class, member, attribute, method, constructor, and package
- Use the access modifiers private and public as appropriate for the guidelines of encapsulation Invoke a method on a particular object
- Use the Java technology API online documentation

Identifiers, Keywords, and Types

- Use comments in a source program
- Distinguish between valid and invalid identifiers
- Use the eight primitive types
- Define literal values for numeric and textual types
- Define the terms primitive variable and reference variable
- Declare variables of class type
- Construct an object using new and describe default initialization
- Describe the significance of a reference variable

Expressions and Flow Control

- Distinguish between instance and local variables
- Describe how to initialize instance variables
- Recognize, describe, and use Java software operators
- Distinguish between legal and illegal assignments of primitive types
- Identify boolean expressions and their requirements in control constructs
- Recognize assignment compatibility and required casts in fundamental types
- Use if, switch, for, while, and do constructions and the labeled forms of break and continue as flow control structures in a progra

Arrays

- Declare and create arrays of primitive, class, or array types Explain why elements of an array are initialized
- Explain how to initialize the elements of an array Determine the number of elements in an array
- Create a multidimensional array
- Write code to copy array values from one array to another

Class Design

- Define inheritance, polymorphism, overloading, overriding, and virtual method invocation
- Use the access modifiers protected and the default (package-friendly)
- Describe the concepts of constructor and method overloading
- Describe the complete object construction and initialization operation

Advanced Class Features

- Create static variables, methods, and initializers
- Create final classes, methods, and variables Create and use enumerated types
- Use the static import statement
- Create abstract classes and methods Create and use an interface

Exceptions and Assertions

- Define exceptions
- Use try, catch, and finally statements
- Describe exception categories
- Identify common exceptions
- Develop programs to handle your own exceptions
- Use assertions
- Distinguish appropriate and inappropriate uses of assertions
- Enable assertions at runtime

Collections and Generics Framework

- Describe the general purpose implementations of the core interfaces in the Collections framework
- Examine the Map interface
- Examine the legacy collection classes
- Create natural and custom ordering by implementing the Comparable and Comparator interfaces
- Use generic collections and type parameters in generic classes
- Refactor existing non-generic code
- Write a program to iterate over a collection
- Examine the enhanced for loop

I/O Fundamentals

- Write a program that uses command-line arguments and system properties
- Examine the Properties class
- Construct node and processing streams, and use them appropriately
- Serialize and deserialize objects
- Distinguish readers and writers from streams, and select appropriately between them

Console I/ O and File I/O

- Read data from the console Write data to the console Describe files and file I/O

Threads

- Define a thread
- Create separate threads in a Java technology program, controlling the code and data that are used by that thread
- Control the execution of a thread and write platform-independent code with threads
- Describe the difficulties that might arise when multiple threads share data
- Use wait and notify to communicate between threads
- Use synchronized to protect data from corruption

Preparation Course For Java Certification 2

Explaining Java Technology

- Describe key concepts of the Java programming language
- List the three Java technology product groups
- Summarize each of the seven stages of the product life cycle

Analyzing a Problem and Designing a Solution

- Analyze a problem using object-oriented analysis
- Design classes from which objects will be created

Developing and Testing a Java Technology Program

- Identify the four components of a class in the Java programming language
- Use the main method in a test class to run a Java technology program from the command line
- Compile and execute a Java technology program

Declaring, Initializing, and Using Variables

- Identify the use the syntax for variables and define the syntax for a variable
- List the eight Java programming language primitive data types

- Declare, initialize, and use variables and constants according to Java programming language guidelines and coding standards
- Modify variable values using operators
- Use promotion and type casting

Creating and Using Objects

- Declare, instantiate, and initialize object reference variables
- Compare how object reference variables are stored in relation to primitive variables
- Use a class (the String class) included in the Java Software Developer Kit (SDK)
- Use the Java 2 Platform, Standard Edition (J2SE) class library specification to learn about other classes in this application programming interface (API)

Using Operators and Decision Constructs

- Identify relational and conditional operators
- Create if and if/else constructs
- Use the switch construct

Using Loop Constructs

- Create while loops
- Develop for loops
- Create do/while loops

Developing and Using Methods

- Describe the advantages of methods and define worker and calling methods
- Declare and invoke a method
- Compare object and static methods
- Use overloaded methods

Implementing Encapsulation and Constructors

- Use encapsulation to protect data
- Create constructors to initialize objects

Creating and Using Arrays

- Code one-dimensional arrays
- Set array values using length attribute and a loop
- Pass arguments to the main method for use in a program
- Create two-dimensional arrays

Implementing Inheritance

- Define and test your use of inheritance
- Explain abstraction
- Explicitly identify class libraries used in your code

Oracle Certified Associate or Java SE 8 Programmer I : 1Z0-808

Java Basics

- Define the scope of variables
- Define the structure of a Java class
- Create executable Java applications with a main method; run a Java program from the command line; including console output.
- Import other Java packages to make them accessible in your code
- Compare and contrast the features and components of Java such as: platform independence, object orientation, encapsulation, etc.

Working With Java Data Types

- Declare and initialize variables (including casting of primitive data types)
- Differentiate between object reference variables and primitive variables
- Know how to read or write to object fields
- Explain an Object's Lifecycle (creation, "dereference by reassignment" and garbage collection)
- Develop code that uses wrapper classes such as Boolean, Double, and Integer.

Using Operators and Decision Constructs

- Use Java operators; including parentheses to override operator precedence
- Test equality between Strings and other objects using == and equals ()
- Create if and if/else and ternary constructs
- Use a switch statement

Creating and Using Arrays

- Declare, instantiate, initialize and use a one-dimensional array
- Declare, instantiate, initialize and use multi-dimensional array

Using Loop Constructs

- Create and use while loops
- Create and use for loops including the enhanced for loop
- Create and use do/while loops
- Compare loop constructs
- Use break and continue

Working with Methods and Encapsulation

- Create methods with arguments and return values; including overloaded methods
- Apply the static keyword to methods and fields
- Create and overload constructors; including impact on default constructors

- Apply access modifiers
- Apply encapsulation principles to a class
- Determine the effect upon object references and primitive values when they are passed into methods that change the values

Working with Inheritance

- Describe inheritance and its benefits
- Develop code that demonstrates the use of polymorphism; including overriding and object type versus reference type
- Determine when casting is necessary
- Use super and this to access objects and constructors
- Use abstract classes and interfaces

Handling Exceptions

- Differentiate among checked exceptions, unchecked exceptions, and Errors
- Create a try-catch block and determine how exceptions alter normal program flow
- Describe the advantages of Exception handling
- Create and invoke a method that throws an exception
- "Recognize common exception classes (such as NullPointerException, ArithmeticExcpetion, ArrayIndexOutOfBoundsException, ClassCastException)"

Working with Selected classes from the Java API

- Manipulate data using the StringBuilder class and its methods
- Creating and manipulating Strings
- Create and manipulate calendar data using classes from java.time.LocalDateTime, java.time.LocalDate, java.time.LocalTime, java.time.format.DateTimeFormatter, java.time.Period
- Declare and use an ArrayList of a given type
- Write a simple Lambda expression that consumes a Lambda Predicate expression

Oracle Certified Professional or Java SE 8 Programmer II - 1Z0-809

Java Class Design

- Implement encapsulation
- Implement inheritance including visibility modifiers and composition
- Implement polymorphism
- Override hashCode, equals, and toString methods from Object class
- Create and use singleton classes and immutable classes

- Develop code that uses static keyword on initialize blocks, variables, methods, and classes

Advanced Java Class Design

- Develop code that uses abstract classes and methods
- Develop code that uses final keyword
- Create inner classes including static inner class, local class, nested class, and anonymous inner class
- Use enumerated types including methods, and constructors in an enum type
- Develop code that declares, implements and/or extends interfaces and use the atOverride annotation.
- Create and use Lambda expressions

Generics and Collections

- Create and use a generic class
- Create and use ArrayList, TreeSet, TreeMap, and ArrayDeque objects
- Use java.util.Comparator and java.lang.Comparable interfaces
- Collections Streams and Filters
- Iterate using forEach methods of Streams and List
- Describe Stream interface and Stream pipeline
- Filter a collection by using lambda expressions
- Use method references with Streams

Lambda Built-in Functional Interfaces

- Use the built-in interfaces included in the java.util.function package such as Predicate, Consumer, Function, and Supplier
- Develop code that uses primitive versions of functional interfaces
- Develop code that uses binary versions of functional interfaces
- Develop code that uses the UnaryOperator interface

Java Stream API

- Develop code to extract data from an object using peek() and map() methods including primitive versions of the map() method
- Search for data by using search methods of the Stream classes including findFirst, findAny, anyMatch, allMatch, noneMatch
- Develop code that uses the Optional class
- Develop code that uses Stream data methods and calculation methods
- Sort a collection using Stream API
- Save results to a collection using the collect method and group/partition data using the Collectors class
- Use flatMap() methods in the Stream API

Exceptions and Assertions

- Use try-catch and throw statements
- Use catch, multi-catch, and finally clauses
- Use Autoclose resources with a try-with-resources statement
- Create custom exceptions and Auto-closeable resources
- Test invariants by using assertions

Use Java SE 8 Date/Time API

- Create and manage date-based and time-based events including a combination of date and time into a single object using LocalDate, LocalTime, LocalDateTime, Instant, Period, and Duration
- Work with dates and times across timezones and manage changes resulting from daylight savings including Format date and times values
- Define and create and manage date-based and time-based events using Instant, Period, Duration, and TemporalUnit

Java I/O Fundamentals

- Read and write data from the console
- Use BufferedReader, BufferedWriter, File, FileReader, FileWriter, FileInputStream, FileOutputStream, ObjectOutputStream, ObjectInputStream, and PrintWriter in the java.iopackage.

Java File I/O (NIO.2)

- Use Path interface to operate on file and directory paths
- Use Files class to check, read, delete, copy, move, manage metadata of a file or directory
- Use Stream API with NIO.2

Java Concurrency

- Create worker threads using Runnable, Callable and use an ExecutorService to concurrently execute tasks
- Identify potential threading problems among deadlock, starvation, livelock, and race conditions
- Use synchronized keyword and java.util.concurrent.atomic package to control the order of thread execution
- Use java.util.concurrent collections and classes including CyclicBarrier and CopyOnWriteArrayList
- Use parallel Fork/Join Framework
- Use parallel Streams including reduction, decomposition, merging processes, pipelines and performance.

Building Database Applications with JDBC

- Describe the interfaces that make up the core of the JDBC API including the Driver, Connection, Statement, and ResultSet interfaces and their relationship to

provider implementations
- Identify the components required to connect to a database using the DriverManager class including the JDBC URL
- Submit queries and read results from the database including creating statements, returning result sets, iterating through the results, and properly closing result sets, statements, and connections

Localization

- Read and set the locale by using the Locale object
- Create and read a Properties file
- Build a resource bundle for each locale and load a resource bundle in an application