98-388
Introduction to Programming Using Java

Exam number: 98-388

Exam title: **MTA 98-388 Introduction to Programming Using Java**

Publish date:

GUID:

Language(s) this exam will be available in:

Audience (IT professionals, Developers, Information workers, etc.): Beginning programming students

Technology: Java 6 SE

Credit type MTA

Exam provider (VUE, Certiport, or both)**:** **Both**

Exam Design

Audience Profile
This is an entry level certification that is intended for application developers working with Java. The MTA exams are targeted at secondary and immediate post-secondary level students of software development, and other entry-level software developers. The code in the 98-388: Introduction to Programming Using Java exam, uses Java SE. The syntax used in this exam is compatible with Java 6 SE through the most recent release.

These Java developers and students require instruction and/or hands-on experience (150 hours) with Java, are familiar with its features and capabilities, and understand how to write, debug and maintain well-formed, well documented Java code.

Skills measured

1. **Understand Java fundamentals**
    1.1. Describe the use of main in a Java application
        *Signature of main, why it is static; how to consume an instance of your own class; command-line arguments
    1.2. Perform basic input and output using standard packages
        *Print statements; importing and using the Scanner class
    1.3. Evaluate the scope of a variable
        *Declaring a variable within a block, class, method

2. **Work with data types, variables, and expressions**
    2.1. Declare and use primitive data type variables
        *Data types include byte, char, int, double, short, long, float, boolean; identify when precision is lost; initialization; how primitives differ from wrapper object types such as Integer and Boolean
    2.2. Construct and evaluate code that manipulates strings
        *String class and string literals, comparisons, concatenation, case and length; String.format methods; string operators; converting a primitive data type to a string; the immutable nature of strings; initialization; null
    2.3. Construct and evaluate code that creates, iterates, and manipulates arrays and array lists
        *One- and two-dimensional arrays, including initialization, null, size, iterating elements, accessing elements; array lists, including adding and removing elements, traversing the list
    2.4. Construct and evaluate code that performs parsing, casting and conversion
        *Implementing code that casts between primitive data types, converts primitive types to equivalent object types, or parses strings to numbers
    2.5. Construct and evaluate arithmetic expressions
        *Arithmetic operators, assignment, compound assignment operators, operator precedence

3. **Implement flow control**
    3.1. Construct and evaluate code that uses branching statements
        *if, else, else if, switch; single-line vs. block; nesting; logical and relational operators

3.2. Construct and evaluate code that uses loops
*while, for, for each, do while; break and continue; nesting; logical, relational, and unary operators

4. Perform object-oriented programming

4.1. Construct and evaluate a class definition
*Constructors; constructor overloading; one class per .java file; this keyword; inheritance and overriding at a basic level

4.2. Declare, implement, and access data members in a class
*private, public, protected; instance data members; static data members; using static final to create constants; describe encapsulation

4.3. Declare, implement, and access methods
*private, public, protected; method parameters; return type; void; return value; instance methods; static methods; overloading

4.4. Instantiate and use a class object in a program
*Instantiation; initialization; null; accessing and modifying data members; accessing methods; accessing and modifying static members; importing packages and classes

5. Compile and debug code

5.1. Troubleshoot syntax errors, logic errors, and runtime errors
*print statement debugging; output from the javac command; analyzing code for logic errors; console exceptions after running the program; evaluating a stack trace

5.2. Implement exception handling
*try catch finally; exception class; exception class types; displaying exception information