Java identifiers

Posted on October 1, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

Q1. Which of these are legal Identifiers in Java?

a) $Ident1
b) _Ident1
c) -Ident1
d) 2Ident1
e) private
f) private1
g) null
h) Ident-1
i) Ident$1
j) \u00A3Ident1
k) \u00A5Ident1

A1. The legal identifiers in Java are are a,b, f, i, j, and k.

Identifiers are names you give to your variables, constants, classes, interfaces and methods. A valid identifier, complying with the following rules:

- The first character of an identifier must be a letter, an underscore(_), or a currency sign(e.g. $).
- The rest of the characters in the identifier can be a letter, underscore, currency sign, or digit. Note that spaces are NOT allowed in identifiers.
- Identifiers are case-sensitive. This means that age and Age are considered as different identifiers.
- Identifiers cannot match any of Java's reserved words like for, int, etc or literals like null, true, and false.

Q2. Can you write a sample code in Java that determines if a given string is a reserved keyword or not?
A2.

```
import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public final class ValidIdentifiers {

    private enum Validity {
        Valid, InvalidIdentifierStart, InvalidId
    };

    private static final String[ ] RESERVED_KEYW
            "continue", "for", "new", "switch",
            "if", "package", "synchronized", "bo
            "private", "this", "break", "double"
            "protected", "throw", "byte", "else"
            "throws", "case", "enum", "instanceo
            "transient", "catch", "extends", "in
            "char", "final", "interface", "stati
            "finally", "long", "strictfp", "vola
            "float", "native", "super", "while"

    private static final String[ ] RESERVED_LITE

    private static Set<String> KEYWORDS = new Ha
            (int)(RESERVED_KEYWORDS.length/0.75)

    private static Set<String> LITERALS = new Ha
            (int)(RESERVED_LITERALS.length/0.75)

    static {
        List<String> list = Arrays.asList(RESERV
        KEYWORDS = new HashSet<String>(list);

        List<String> listLit = Arrays.asList(RES
        LITERALS = new HashSet<String>(listLit);
    }

    public static final Validity valid(String in
        if (input.length( ) == 0
                || !Character.isJavaIdentifierSt
            return Validity.InvalidIdentifierSta
        }

        for (int i = 1; i < input.length( ); i++
            if (!Character.isJavaIdentifierPart(
                return Validity.InvalidIdentie
            }
        }

        if (KEYWORDS.contains(input)) {
            return Validity.ReservedKeyWord;
        }

        if (LITERALS.contains(input)) {
            return Validity.ReservedLiteral;
        }
        return Validity.Valid;
    }
}
```

Note: true, false, and null might seem like keywords, but they are actually literals; you cannot use them as identifiers in your programs.

Note: **const** and **goto** are reserved, but not currently used. **enum** was added in Java 5. **strictfp** allows you to have more predictable control over floating-point arithmetic.

Q3. Can you talk us through the code highlighting some of the key language and design features?
A3.

- Use of enums and generics indicates that this code must be using JDK version 1.5 or greater.
- **Private** access modifiers are used where required to encapsulate the internal details.
- The class is marked final so that it cannot be extended.
- It follows the “code to interface” design principle. For example, the Set, List, etc shown below are interfaces.

```
private static Set<String> KEYWORDS = new H
//...
```

```
List<String> list = Arrays.asList(RESERVED_

```

- Making use of the **Java API methods** where possible. For example, the methods shown below from the Character and Arrays classes, simplify your code. So don't memorize the Java API, but keep it handy and constantly refer to it.

```
Character.isJavaIdentifierStart(input.charA
Character.isJavaIdentifierPart(input.charAt
List<String> list = Arrays.asList(RESERVED_
```

- If a size of a collection is known in advance, it is a best practice to set its initial size appropriately to prevent any resizing. Implementing the code as shown below would not quite work.

```
private static Set<String> KEYWORDS = new H
```

The internal threshold for HashSets and HashMaps are calculated as (int) (capacity * loadFactor). The default loadFactor is 0.75. This means the HashSet will resize after 75% of its capacity has been reached. The resizing and rehashing of the set can be prevented as follows,

```
private static Set<String> KEYWORDS = new H
```

- Checking for null and empty string as a precondition in the beginning of the valid(String input) method.

```
if (input == null || input.length( ) == 0)
```

The above code snippet is a slight deviation of the **fail fast principle**, which states that check and report any possible failures as soon as possible. Testing your code for failure points will lead to better, safer, and more reliable code.

Q4. Do you have any recommendations to further improve the code?

A4. Yes.

1) The **Apache commons library** class *StringUtils* can be introduced here. The method *isEmpty (String input)* can be used to check for both null and empty string. This library does have other useful methods that can be used elsewhere in the application code to enforce the fail fast principle.

2) The RESERVED_KEYWORDS and RESERVED_LITERALS constants may be loaded from a configuration file. This will ensure that if new keywords or literals are added to Java in the future, it will require only a configuration change.

Q5. What is an escape character? Can you list some character escape codes in Java? What are the differences between decimal, octal, and hexadecimal literals?

A5. The **escape character**, the back slash \, is the character that signals the following character is not what it normally means. Java provides escape sequences for several non-graphical characters. All characters can be specified as a hexadecimal Unicode character (\udddd) with some as an octal character (\ddd where the first d is limited to 0-3, and the others 0-7).

```
char c1 = 'A';                   //Prints 'A',
 char c2 = '\n';                  //Prints new
 char c3 = '\u0041';              //Prints 'A',
 char c4 = 65;                    //Prints 'A'

```

Q6. How does Java treat Unicode escapes within string literals?

A6.

```
System.out.println("\u00A3-Pound, \u00A5-Yen");
```

Q7. Why is it a best practice to adhere to naming conventions when naming the variables, classes, interfaces, etc?

A7. **Naming conventions** make programs more understandable by making them easier to read. They can also give information about the function of the identifier. For example, whether it's a constant, package, or class.

- For classes and interfaces, use **UpperCamelCase**.
- For class members and local variables use **lowerCamelCase**
- For packages, use reverse URI, e.g. org.project.subsystem
- For constants, use ALL_CAPS.
-

If you really want to begin a variable name with a digit, prefix the name you'd like to have (e.g. 9pins) with an underscore, e.g. _9pins. Otherwise use something like ninePins.

Some try to differentiate between member variables, local variables, and arguments with different patterns. For example,

— field (aka instance or member variable): **_name** or **this**.name. "this" is a reserved keyword in Java referring to the current object.
— argument : **aName**
— parameter : **pName**
— local variable : **_name**

There is no real objective reason to prefer one style over the other as long as it is consistent across the team or organization. **Checkstyle** is an open-source tool that can help enforce coding standards and best practices.

Popular Posts

- ♦ 11 Spring boot interview questions & answers

  857 views
- ♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

  825 views

18 Java scenarios based interview Questions and Answers

447 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

400 views

♦ 7 Java debugging interview questions & answers

311 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

301 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

292 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

286 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

263 views

8 Git Source control system interview questions & answers

215 views

Bio | **Latest Posts**

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.

‹ Flyweight pattern and improve memory usage & performance

01: Java data types interview Q&A ›

Posted in member-paid, Reserved Key Words

Leave a Reply

Logged in as geethika. Log out?

Comment

Post Comment

08: Hibernate au
09: Hibernate en
10: Spring, Java
11: Hibernate de
12: Hibernate cu
AngularJS (2)
♥ 8 AngularJS in
More Angular JS
Git & SVN (6)
♥ Git & Maven fo
♥ Merging Vs rel
♥ Understanding
6 more Git interv
8 Git Source cor
Setting up Cygw
JMeter (2)
♥ JMeter for test
♦ JMeter perform
JSF (2)
JSF interview Q&
More JSF intervi
Maven (3)
♥ Git & Maven fo
12 Maven intervi
7 More Maven ir
Testing & Profiling/Sa
Automation Testing
♥ Selenium and
Code Coverage (2)
Jacoco for unit te
Maven and Cobe
Code Quality (2)
♥ 30+ Java Code
♦ Ensuring code
jvisualvm profiling (
01: jvisualvm to
02: jvisualvm to
03: jvisualvm to
Performance Testir
♥ JMeter for test
♦ JMeter perform

Unit Testing Q&A (2
BDD Testing (4)
Java BDD (Be
jBehave and E
jBehave and j
jBehave with t
Data Access Uni
♥ Unit Testing
Part #3: JPA H
Unit Test Hibe
Unit Test Hibe
JUnit Mockito Sp
JUnit Mockito
Spring Con
Unit Testing
Part 1: Unit te
Part 2: Mockit
Part 3: Mockit
Part 4: Mockit
Part 5: Mockit
Testing Spring T
Integration Un
Unit testing Sp
♦ 5 Java unit tes
JUnit with Hamc
Spring Boot in ui
Other Interview Q&A f
Finance Domain In
12+ FX or Forex
15 Banking & fin
FIX Interview Q&A
20+ FIX basics ii
Finding your way
Groovy Interview Q
Groovy Coding (
Cash balance
Sum grades G
♥ Q1 – Q5 Groov
♦ 20 Groovy clos
♦ 9 Groovy meta
Groovy method (

As a Java Architect

Java architecture & design concepts interview Q&As with diagrams | What should be a typical Java EE architecture?

Senior Java developers must have a good handle on

80+ step by step Java Tutorials

Preparing for Java written & coding tests

How good are your...to go places?

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

🔆 Turn readers of your Java CV go from “Blah blah” to “Wow”? 🔆 How to prepare for Java job interviews? 🔆 16 Technical Key Areas 🔆 How to choose from multiple Java job offers?

Select Category ▼

© Disclaimer