Register | Login | Logout | Contact Us

Java-Success.com

Industrial strength Java/JEE Career Companion for those who want to go places

search here … Go

Home | Java FAQs | 600+ Java Q&As | Career | Tutorials | Member | Why?

Can u Debug? | Java 8 ready? | Top X | Productivity Tools | Judging Experience?

Home › Interview › Core Java Interview Q&A › Generics › ♥ Overloaded methods Vs Generic methods in Java with JD-GUI & javap to look under the covers

♥ Overloaded methods Vs Generic methods in Java with JD-GUI & javap to look under the covers

Posted on December 15, 2015 by Arulkumaran Kumaraswamipillai

In an earlier post we looked at Understanding Overriding, Hiding, and Overloading in Java?. We also discussed how “method overriding” gives **polymorphism**. In this post, let’s see how a “generic method” can replace a number of overloaded methods. Then see under the hood as to what happens to the compiled code.

Overloaded methods example

9 tips to earn more | What can u do to go places? | **945+** members. LinkedIn Group. **Reviews**

600+ Full Stack Java/JEE Interview Q&As ♥Free ♦FAQs

open all | close all

⊞ Ice Breaker Interview
⊟ Core Java Interview (
⊞ Java Overview (4)
⊞ Data types (6)
⊞ constructors-methc
⊞ Reserved Key Wor
⊞ Classes (3)
⊞ Objects (8)
⊞ OOP (10)
⊞ GC (2)
⊟ Generics (5)

3 overloaded methods to print 3 different types of data. If you notice it carefully, in all 3 methods the method signatures and the actual method body are the same except for the data types they work with like Integer, Character, and String.

```
package test;

public class OverLoadedMethods {

    public static void main(String[] args) {
        Integer[] numbers = new Integer[] { 1, 2
        Character[] characters = new Character[]
        String[] strings = new String[] { "AB",

        print(numbers);
        System.out.println();
        print(characters);
        System.out.println();
        print(strings);
    }

    private static void print(Integer[] input) {
        for (Integer in : input) {
            System.out.printf("%s ", in);
        }
    }

    private static void print(Character[] input)
        for (Character in : input) {
            System.out.printf("%s ", in);

        }
    }

    private static void print(String[] input) {
        for (String in : input) {
            System.out.printf("%s ", in);
        }
    }

}
```

Output:

```
1 2 3 4
A B C
AB BC CD
```

Generic method example

1 generic method to replace 3 overloaded methods. It takes a generic type "T"

♥ Java Generics
♥ Overloaded m
♦ 12 Java Gener
♦ 7 rules to reme
3 scenarios to ge
FP (8)
IO (7)
Multithreading (12)
Algorithms (5)
Annotations (2)
Collection and Data
Differences Betwee
Event Driven Progr
Exceptions (2)
Java 7 (2)
Java 8 (24)
JVM (6)
Reactive Programn
Swing & AWT (2)
JEE Interview Q&A (3
Pressed for time? Jav
SQL, XML, UML, JSC
Hadoop & BigData Int
Java Architecture Inte
Scala Interview Q&As
Spring, Hibernate, & I
Testing & Profiling/Sa
Other Interview Q&A f
▶ Free Java Interview

As a Java Architect

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

```

package test;

public class GenericMethod<T> {

    public static void main(String[] args) {
        Integer[] numbers = new Integer[] { 1, 2
        Character[] characters = new Character[]
        String[] strings = new String[] { "AB",

        print(numbers);
        System.out.println();
        print(characters);
        System.out.println();
        print(strings);
    }

    //one generic method instead of 3 overloaded
    private static <T>  void print(T[] input) {
        for (T in : input) {
            System.out.printf("%s ", in);
        }
    }

}

```

Output:

```
1 2 3 4
A B C
AB BC CD
```

What is happening under the covers?

Isn't Java Generics a **compile-time** phenomenon? Yes. Let's dig a bit deeper with two handy tools **1)** JD-GUI (Java Decompiler GUI) **2)** javap.

JD-GUI

is used to de-compile a Java class file. open the JD-GUI, and drag and drop the **GenericMethod.class**. As you can see an Object[] is used to handle different types. The generic type "T" in the method body is **erased by the compiler**.

```

package test;
```

Senior Java developers must have a good handle on

open all | close all

⊞ Best Practice (6)
⊞ Coding (26)
⊞ Concurrency (6)
⊞ Design Concepts (7)
⊞ Design Patterns (11)
⊞ Exception Handling (3
⊞ Java Debugging (21)
⊞ Judging Experience I
⊞ Low Latency (7)
⊞ Memory Managemen
⊞ Performance (13)
⊞ QoS (8)
⊞ Scalability (4)
⊞ SDLC (6)
⊞ Security (13)
⊞ Transaction Managen

80+ step by step Java Tutorials

open all | close all

⊞ Setting up Tutorial (6)
⊞ Tutorial - Diagnosis (2
⊞ Akka Tutorial (9)
⊞ Core Java Tutorials (2
⊞ Hadoop & Spark Tuto
⊞ JEE Tutorials (19)
⊞ Scala Tutorials (1)
⊞ Spring & HIbernate T
⊞ Tools Tutorials (19)
⊞ Other Tutorials (45)

```

import java.io.PrintStream;

public class GenericMethod<T>
{
  public static void main(String[] args)
  {
    Integer[] numbers = { Integer.valueOf(1), In
    Character[] characters = { Character.valueOf
    String[] strings = { "AB", "BC", "CD" };

    print(numbers);
    System.out.println();
    print(characters);
    System.out.println();
    print(strings);
  }

  private static <T> void print(T[] input)
  {
    Object[] arrayOfObject = input;
    int j = input.length;
    for (int i = 0; i < j; i++) {
       Object in = arrayOfObject[i];
       System.out.printf("%s ", new Object[] { i
    }
  }
}


```

javap command

```


javap -c GenericMethod.class
Compiled from "GenericMethod.java"
public class test.GenericMethod<T> {
  public test.GenericMethod();
    Code:
       0: aload_0
       1: invokespecial #8                  // M
       4: return

  public static void main(java.lang.String[]);
    Code:
       0: iconst_4
       1: anewarray     #18                 // c
       4: dup
       5: iconst_0
       6: iconst_1
       7: invokestatic  #20                 // M
      10: aastore
      11: dup
      12: iconst_1
      13: iconst_2
      14: invokestatic  #20                 // M
      17: aastore
      18: dup
      19: iconst_2
      20: iconst_3
      21: invokestatic  #20                 // M
      24: aastore
```

Preparing for Java written & coding tests

open all | close all

⊞ ♦ Complete the given
⊞ Can you write code? (
⊞ Converting from A to I
⊞ Designing your classe
⊞ Java Data Structures
⊞ Passing the unit tests
⊞ What is wrong with th
⊞ Writing Code Home A
⊞ Written Test Core Jav
⊞ Written Test JEE (1)

How good are your...to go places?

open all | close all

⊞ Career Making Know-
⊞ Job Hunting & Resum

```
      25: dup
      26: iconst_3
      27: iconst_4
      28: invokestatic  #20                 // M
      31: aastore
      32: astore_1
      33: iconst_3
      34: anewarray     #24                 // c
      37: dup
      38: iconst_0
      39: bipush        65
      41: invokestatic  #26                 // M
      44: aastore
      45: dup
      46: iconst_1
      47: bipush        66
      49: invokestatic  #26                 // M
      52: aastore
      53: dup
      54: iconst_2
      55: bipush        67
      57: invokestatic  #26                 // M
      60: aastore
      61: astore_2
      62: iconst_3
      63: anewarray     #29                 // c
      66: dup
      67: iconst_0
      68: ldc           #31                 // S
      70: aastore
      71: dup
      72: iconst_1
      73: ldc           #33                 // S
      75: aastore
      76: dup
      77: iconst_2
      78: ldc           #35                 // S
      80: aastore
      81: astore_3
      82: aload_1
      83: invokestatic  #37                 // M
      86: getstatic     #41                 // F
      89: invokevirtual #47                 // M
      92: aload_2
      93: invokestatic  #37                 // M
      96: getstatic     #41                 // F
      99: invokevirtual #47                 // M
     102: aload_3
     103: invokestatic  #37                 // M
     106: return
}


```

Note lines from **83**. The **#37** shows that “print” method is invoked with “java.lang.Object” and “**V**” means “void”. “**L**” means Object, and “**[**” means an array.

```

Method print:([Ljava/lang/Object;)V

```

```

Type         Chararacter
------------------------------------------------
boolean           Z
byte              B
char              C
double            D
float             F
int               I
long              J
object            L
short             S
void              V
array             [

```

javap with flag "-l" to relate to original code

```

javap -c -l GenericMethod.class
Compiled from "GenericMethod.java"
public class test.GenericMethod<T> {
  public test.GenericMethod();
    Code:
       0: aload_0
       1: invokespecial #8                  //
       4: return
    LineNumberTable:
      line 3: 0
    LocalVariableTable:
      Start  Length  Slot  Name   Signature
          0       5     0  this   Ltest/Generic

  public static void main(java.lang.String[]);
    Code:
       0: iconst_4
       1: anewarray     #18                 //
       4: dup
       5: iconst_0
       6: iconst_1
       7: invokestatic  #20                 //
      10: aastore
      11: dup
      12: iconst_1
      13: iconst_2
      14: invokestatic  #20                 //
      17: aastore
      18: dup
      19: iconst_2
      20: iconst_3
      21: invokestatic  #20                 //
      24: aastore
      25: dup
      26: iconst_3
      27: iconst_4
      28: invokestatic  #20                 //
      31: aastore
      32: astore_1
      33: iconst_3
      34: anewarray     #24                 //
      37: dup
      38: iconst_0
      39: bipush        65
      41: invokestatic  #26                 //
      44: aastore
      45: dup
      46: iconst_1
      47: bipush        66
      49: invokestatic  #26                 //
      52: aastore
      53: dup
      54: iconst_2
      55: bipush        67
      57: invokestatic  #26                 //
      60: aastore
      61: astore_2
      62: iconst_3
      63: anewarray     #29                 //
      66: dup
      67: iconst_0
      68: ldc           #31                 //
      70: aastore
      71: dup
      72: iconst_1
      73: ldc           #33                 //
      75: aastore
      76: dup
      77: iconst_2
      78: ldc           #35                 //
      80: aastore
      81: astore_3
      82: aload_1
      83: invokestatic  #37                 //
      86: getstatic     #41                 //
      89: invokevirtual #47                 //
      92: aload_2
      93: invokestatic  #37                 //
      96: getstatic     #41                 //
      99: invokevirtual #47                 //
     102: aload_3
     103: invokestatic  #37                 //
     106: return
    LineNumberTable:
      line 6: 0
      line 7: 33
      line 8: 62
      line 10: 82
      line 11: 86
      line 12: 92
      line 13: 96
      line 14: 102
      line 15: 106
    LocalVariableTable:
      Start  Length  Slot  Name   Signature
          0     107     0  args   [Ljava/lang/S
         33      74     1 numbers   [Ljava/lang
         62      45     2 characters   [Ljava/l
         82      25     3 strings   [Ljava/lang
}


```

Look at the “**LineNumberTable**“. For example:

The **line 6** in the source code

```
Integer[] numbers = new Integer[] { 1, 2, 3, 4 };
```

corresponds to dessembeld **line 0**:

```
0: iconst_4
```

You may also like: javap for debugging and better understanding some Java concepts with 3 practical examples

Popular Posts

- ♦ 11 Spring boot interview questions & answers

  861 views

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

  829 views

- 18 Java scenarios based interview Questions and Answers

  448 views

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

  407 views

- ♦ 7 Java debugging interview questions & answers

  311 views

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

  303 views

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

  294 views

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

  288 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.

‹ ▶ Beginner Java Object class interview Q&A

Reloading configuration files in Java without stopping the server ›

Posted in Generics

Tags: Free Content

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

The contents in this Java-Success are copy righted. The author has the right to correct or enhance the current content without any prior notice.

These are general advice only, and one needs to take his/her own circumstances into consideration. The author will not be held liable for any damages caused or alleged to be caused either directly or indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. No guarantees are made regarding the accuracy or usefulness of content, though I do make an effort to be accurate. Links to external sites do not imply endorsement of the linked-to sites.

© 2016 Java-Success.com ↑ Responsive Theme powered by WordPress