*OCP Java SE 7 Programmer II*

*7. Assertions and Java 7 Exceptions*

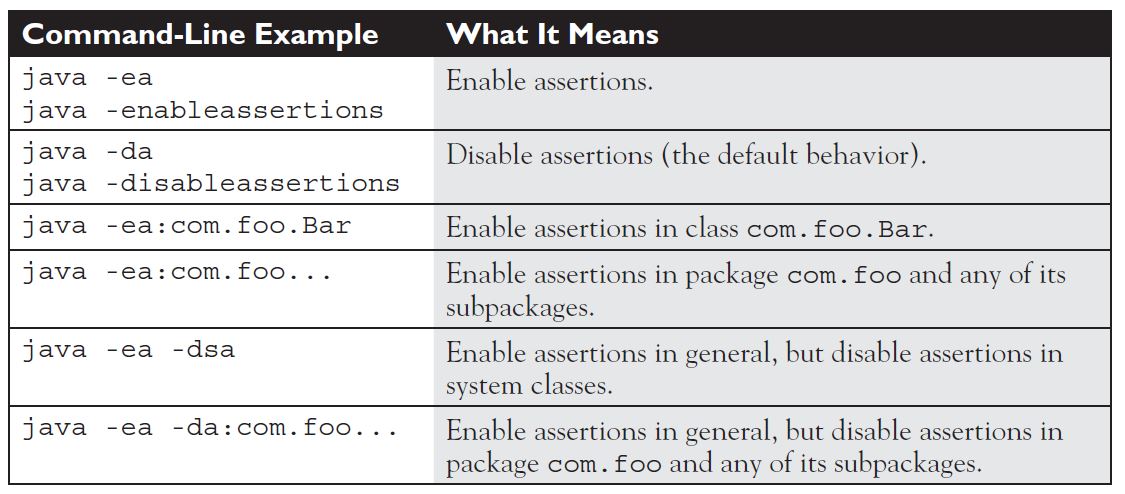
***Assertions:***

Prior to java 1.4 assert is not a keyword in java, so we can use as identifier also.

int assert = getInitialValue();//**compiles.**

From java 1.4 onwards assert is keyword, so we can’t use as identifier.

int assert = getInitialValue();//**not compiles.**



* Don’t Use Assertions to Validate Arguments to a public Method.
* Don't Use Assertions to Validate Command-Line Arguments.
* Do Use Assertions to Validate Arguments to a private Method.
* Do Use Assertions, Even in public Methods, to Check for Cases That You Know Are Never, Ever Supposed to Happen.

switch(x) {

case 1: y = 3; break;

case 2: y = 9; break;

case 3: y = 27; break;

default: assert false; // we're never supposed to get here!

}

* Don't Use assert Expressions That Can Cause Side Effects!

The following would be a very bad idea:

public void doStuff() {

assert (modifyThings());

// continues on

}

public boolean modifyThings() {

y = x++;

return true;

}

assert expressions aren't guaranteed to always run, so you don't want your code to behave differently depending on whether assertions are enabled.

***Working with Java 7 Exception Handling***

You can't use the variable name multiple times in a multi-catch. The following won't compile:

catch(Exception1 e1 | Exception2 e2)3

With multi-catch, order doesn’t matter. The following two snippets are equivalent to each other:

catch(SQLException | IOException e) // these two statements are equivalent

catch(IOException | SQLException e)

With multi-catch, you have to make sure a given exception can only match one type. The following will not compile:

catch(FileNotFoundException | IOException e)

catch(IOException | FileNotFoundException e)

You'll get a compiler error that looks something like:

The exception FileNotFoundException is already caught by the alternative IOException

Remember, multi-catch is only for exceptions in different inheritance hierarchies. To make sure this is clear, what do you think happens with the following code:

catch(IOException | Exception e)

That’s right. It won’t compile because IOException is a subclass of Exception. Which means it is redundant and the compiler won’t accept it.

catch (SQLException | IOException e) {

e = new IOException();// **won’t compile.**

}

Since multi-catch uses multiple types, there isn't a clearly defined type for the variable that you can set. Java solves this by making the catch parameter final when that happens.

catch (SQLException | IOException e) {

log(e);

throw e;

}

Lucky for us, Java 7 helps us out here as well with a new feature.

catch (Exception e) {

log(e);

throw e;

}

In Java 7, } catch (Exception e) { doesn't really catch ANY Exception subclass. The code may say that, but the compiler is translating for you. The compiler says, "Well, I know it can't be just any exception because the throws clause won't let me. I'll pretend the developer meant to only catch SQLException and IOException. After all, if any others show up, I'll just fail compilation on throw e; just like I used to in Java 6." Tricky, isn't it?

*8. String Processing, Data Formatting, Resource Bundles*

Date d1 = new Date(1\_000\_000\_000\_000L);

System.out.println("1st date " + d1.toString());

Calendar c = Calendar.getInstance();

c.setTime(d1); // #1

if(Calendar.SUNDAY == c.getFirstDayOfWeek()) // #2

System.out.println("Sunday is the first day of the week");

System.out.println("trillionth milli day of week is " + c.get(Calendar.DAY\_OF\_WEEK)); // #3

c.add(Calendar.MONTH, 1); // #4

Date d2 = c.getTime(); // #5

System.out.println("new date " + d2.toString() );

**Output:**

1st date Sat Sep 08 19:46:40 MDT 2001

Sunday is the first day of the week

trillionth milli day of week is 7

new date Mon Oct 08 19:46:40 MDT 2001

The other Calendar method you should know for the exam is the roll() method. The roll() method acts like the add() method, except that when a part of a Date gets incremented or decremented, larger parts of the Date will not get incremented or decremented. Hmmm… for instance:

// assume c is October 8, 2001

c.roll(Calendar.MONTH, 9); // notice the year in the output

Date d4 = c.getTime();

System.out.println("new date " + d4.toString() );

**The output would be something like this:**

new date Fri Jul 08 19:46:40 MDT 2001

Notice that the year did not change, even though we added nine months to an October date. In a similar fashion, invoking roll() with HOUR won't change the date, the month, or the year.

DateFormat.format() to convert Date into a String.

DateFormat.parse() to convert String into a Date.

