Feel free to contact santhosh_arena@yahoo.co.in

SCJP java Programmer certification Exam And Training. This site is entirely independent of Sun Microsystems Inc, The Sun Certified Java Programmers Exam (SCJP) is the internationally recognized java certification exam. I am offering free mock Exam preparation questions, practice tests, tutorials, certification faq and sample code. This page contains a very detailed tutorial organized by topic. At the end of the tutorial you can learn from your mistakes and apply your concepts on the free mock exams java certification practice tests.

Chapter 4 : Converting and Casting

Unary Numeric Promotion

Contexts:

a) Operand of the unary arithmetic operators + and -

b) Operand of the unary integer bit-wise complement operator ~

c) During array creation, for example new int[x], where the dimension expression x must evaluate to an int value.

d) Indexing array elements, for example table['a'], where the index expression must evaluate to an int value.

e) Individual operands of the shift operators.

Binary numeric promotion

Contexts:

a) Operands of arithmetic operators *, / , %, + and -

b) Operands of relational operators <, <= , > and >=

c) Numeric Operands of equality operators == and !=

d) Integer Operands of bit-wise operators &, ^ and |

Conversion of Primitives

1. 3 types of conversion - assignment conversion, method call conversion and arithmetic promotion

2. boolean may not be converted to/from any non-boolean type.

3. Widening conversions accepted. Narrowing conversions rejected.

4. byte, short can't be converted to char and vice versa. (but can be cast)

5. Arithmetic promotion

5.1 Unary operators

a) if the operand is byte, short or char {

convert it to int;

}

else {

do nothing; no conversion needed;

}

5.2 Binary operators

a) if one operand is double {

all double; convert the other operand to double;

}

else if one operand is float {

all float; convert the other operand to float;

}

else if one operand is long {

all long; convert the other operand to long;

}

else {

all int; convert all to int;

}

6. When assigning a literal value to a variable, the range of the variable's data type is checked against the value of the literal and assignment is allowed or compiler will produce an error.

char c = 3; // this will compile, even though a numeric literal is by default an int since the range of char will accept the value

int a = 3;

char d = a; // this won't compile, since we're assigning an int to char

char e = -1; // this also won't compile, since the value is not in the range of char

float f = 1.3; // this won't compile, even though the value is within float range. Here range is not important, but precision is. 1.3 is by default a double, so a specific cast or f = 1.3f will work.

float f = 1/3; // this will compile, since RHS evaluates to an int.

Float f = 1.0 / 3.0; // this won't compile, since RHS evaluates to a double.

7. Also when assigning a final variable to a variable, even if the final variable's data type is wider than the variable, if the value is within the range of the variable an implicit conversion is done.

byte b;

final int a = 10;

b = a; // Legal, since value of 'a' is determinable and within range of b

final int x = a;

b = x; // Legal, since value of 'x' is determinable and within range of b

int y;

final int z = y;

b = z; // Illegal, since value of 'z' is not determinable

8. Method call conversions always look for the exact data type or a wider one in the method signatures. They will not do narrowing conversions to resolve methods, instead we will get a compile error.

Here is the figure of allowable primitive conversion.

byte → short → int → long → float → double

↑

char

[Casting of Primitives]

9. Needed with narrowing conversions. Use with care - radical information loss. Also can be used with widening conversions, to improve the clarity of the code.

10. Can cast any non-boolean type to another non-boolean type.

11. Cannot cast a boolean or to a boolean type.

Conversion of Object references

12. Three types of reference variables to denote objects - class, interface or array type.

13. Two kinds of objects can be created - class or array.

14. Two types of conversion - assignment and method call.

15. Permitted if the direction of the conversion is 'up' the inheritance hierarchy. Means that types can be assigned/substituted to only super-types - super-classes or interfaces. Not the other way around, explicit casting is needed for that.

16. Interfaces can be used as types when declaring variables, so they participate in the object reference conversion. But we cannot instantiate an interface, since it is abstract and doesn't provide any implementation. These variables can be used to hold objects of classes that implement the interface. The reason for having interfaces as types may be, I think, several unrelated classes may implement the same interface and if there's a need to deal with them collectively one way of treating them may be an array of the interface type that they implement.

17. Primitive arrays can be converted to only the arrays of the same primitive type. They cannot be converted to another type of primitive array. Only object reference arrays can be converted / cast.

18. Primitive arrays can be converted to an Object reference, but not to an Object[] reference. This is because all arrays (primitive arrays and Object[]) are extended from Object.

Casting of Object references

19. Allows super-types to be assigned to subtypes. Extensive checks done both at compile and runtime. At compile time, class of the object may not be known, so at runtime if checks fail, a ClassCastException is thrown.

20. Cast operator, instanceof operator and the == operator behave the same way in allowing references to be the operands of them. **You cannot cast or apply instanceof or compare unrelated references, sibling references or any incompatible references.**

Compile-time Rules

- When old and new types are classes, one class must be the sub-class of the other.

a) When old and new types are arrays, both must contain reference types and it must be legal to cast between those types (primitive arrays cannot be cast, conversion possible only between same type of primitive arrays).

b) We can always cast between an interface and a non-final object.

Run-time rules

a) If new type is a class, the class of the expression being converted must be new type or extend new type.

b) If new type is an interface, the class of the expression being converted must implement the interface.

An Object reference can be converted to: (java.lang.Object)

a) an Object reference

b) a Cloneable interface reference, with casting, with runtime check

c) any class reference, with casting, with runtime check

d) any array referenece, with casting, with runtime check

e) any interface reference, with casting, with runtime check

A Class type reference can be converted to:

a) any super-class type reference, (including Object)

b) any sub-class type reference, with casting, with runtime check

c) an interface reference, if the class implements that interface

d) any interface reference, with casting, with runtime check (except if the class is final and doesn't implement the interface)

An Interface reference can be converted to:

a) an Object reference

b) a super-interface reference

c) any interface/class reference with casting, with runtime check (except if the class is final and doesn't implement the interface)

A Primitive Array reference can be converted to:

a) an Object reference

b) a Cloneable interface reference

c) a primitive array reference of the same type

An Object Array reference can be converted to:

a) an Object reference

b) a Cloneable interface reference

c) a super-class Array reference, including an Object Array reference

d) any sub-class Array reference with casting, with runtime check