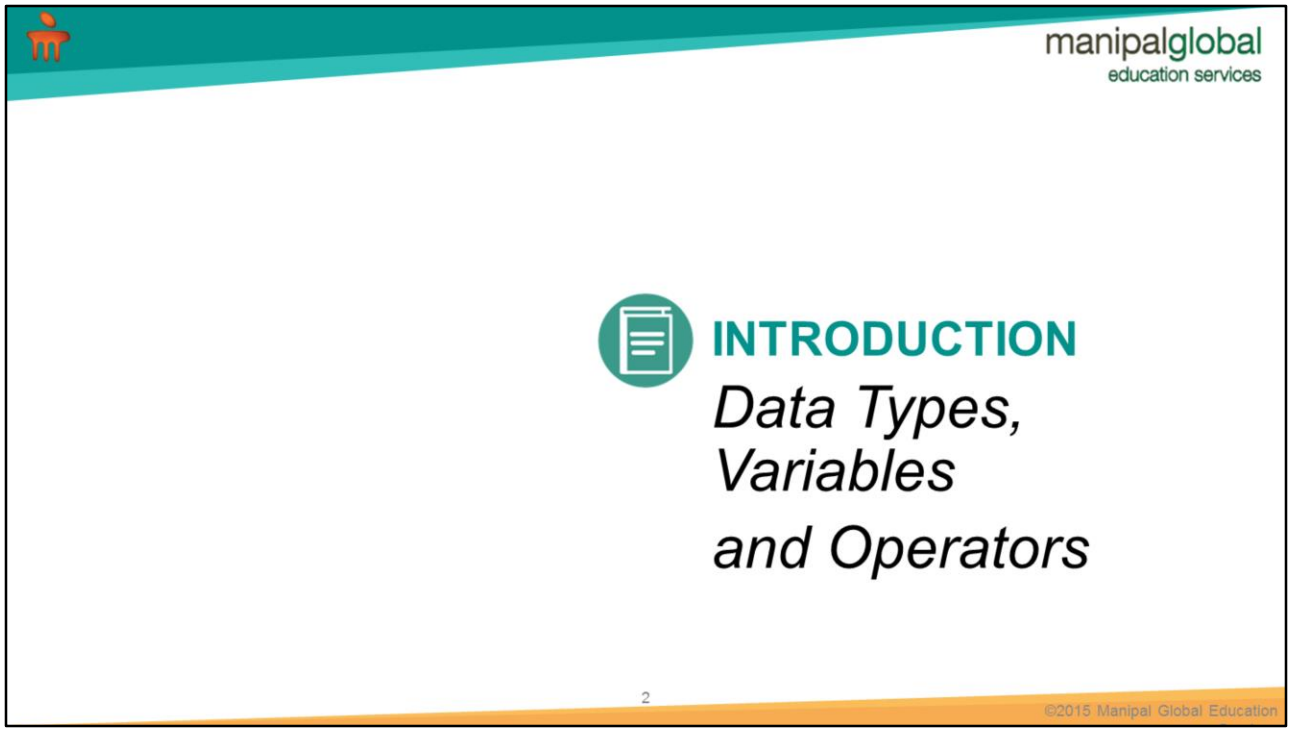
Java Language Fundamentals

- Data types, variables and operators


manipalglobal
education services
INTRODUCTION
Data Types,
Variables
and Operators
2
©2015 Manipal Global Education

LEARNING OBJECTIVES

At the end of this lesson, you will be able to:

- Understand variables
- Explain different data types in Java
- List and use different types of operators
- Demonstrate type casting

Refer package **com.mgait.fundamentals** in the provided code base for demo programs on the topics covered in this presentation

manipalglobal
education services

CONCEPT

Elements of a class

manipalglobal
education services

BASIC ELEMENTS OF JAVA CLASS

```
public class Order {

    int orderId ;
    int orderQty;

    public static void main(String[] args) {
        Order o1 = new Order();
        System.out.println(o1.orderId);
    }
}
```

Class Order.java

Statement

Statement Block

Statement

- Combination of keywords and variables
- End with a semicolon
- Provide an instruction to the Java interpreter

Statement block

- Consists of a number of statement's enclosed in curly brackets
- Can be nested within another block

manipalglobal
education services

COMMENTS

- Provide information about the code
- Helps in easily understanding and maintaining code
- Ignored by the java compiler

Single Line Comment - //

```
// Prints value of OrderId
```

Multiline Comment - /* */

```
/* Below Code is used to
   calculate simple interest */
```

- javadoc comments are used by Javadoc tool for creating Java documentation webpages

Javadoc comment - / */**

```
/**
   * Method setQty is used to set the order quantity
 */
```

IDENTIFIERS

- Used to uniquely identify variables, methods, classes etc
- Rules
  - Case sensitive
  - Can only begin with a alphabet, "$", "_" | **ex: $price, _name**
  - Can contain alphanumeric characters
  - Should not be java keyword | **ex : public, class, float, int**
- Naming Conventions
  - Class name should follow PascalCase | **ex : Order, Account**
  - Methods and variables should follow camelCase | **ex : deposit(), getDetails()**
  - Use full words instead of abbreviations

KEYWORDS

- Keywords are reserved words that are predefined in the language
- Cannot use keywords as identifiers

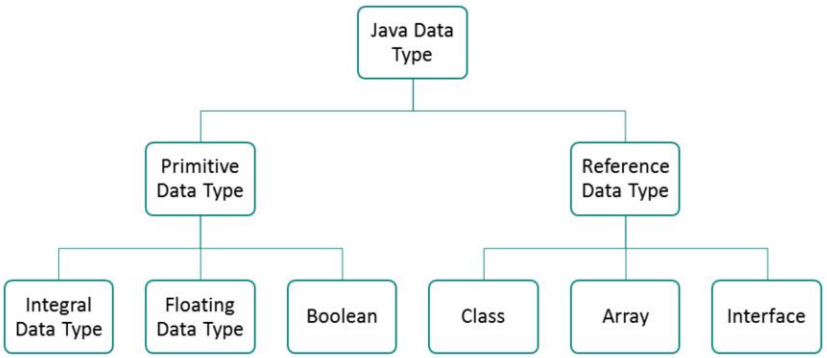
abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

CONCEPT

Data Types and Variables

manipalglobal
education services

DATA TYPES



- **Primitive data type** are predefined data types, which hold the value of the declared data type
- **Reference data type** is a data type that is used to store the reference (address) of an object

manipalglobal
education services

PRIMITIVE DATA TYPES

Type	Bit depth	Value range	Default value
byte	8	-128 to 127	0
short	16	-32,768 to 32,767	0
int	32	(-2 ^ 31) to (2 ^ 31 – 1)	0
long	64	(-2 ^ 63) to (2 ^ 63 – 1)	0L
float	32	Varies	0.0f
double	64	varies	0.0d
boolean	Depends on JVM	True or false	false
char	16 unsigned Unicode character	0 to 65535	0

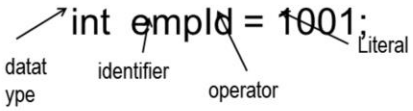
Integral

Floating

Boolean

VARIABLE

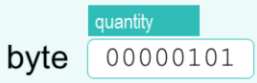
- Basic unit of storage in a Java program
- Container that holds data and can be used throughout a program
- Defined by the combination of a data type, an identifier and an optional initializer



manipalglobal
education services

VARIABLE DECLARATION

```
byte  quantity = 5;
```




```
short  quantity = 5;
```


```
int  count = 10;
long noOfEmployees = 1000L;   // long literal has to suffixed with 'l' or 'L'
int  hexCount = 0x1F;         // hex literal starts with 0x
int  octCount = 011;          // octal literal starts with 0
int  octCount = 10_000.00;    // numeric literals can have _ as separators
```

Default value for all the integer literals is integer

manipalglobal
education services

VARIABLE DECLARATION

```
float price = 1208.976f;    // Float literal has to suffixed with 'f' or 'F'
double price = 2000.25;
double price = 2000.25d;
double range = 2e+05;
```

Default value for all floating-point literals is double

```
char gender = 'M';     // character literals should be within single quotes
char space = '\u0020';

boolean result = true; // valid boolean literal are true or false
```

FINAL VARIABLES

- Variable can be declared as **final**
- Final variables cannot be changed, once initialized
- Common coding convention is to choose uppercase letters for static and final variable

```
final int BUF_SIZE = 1024;
```

REFERENCE DATA TYPES

- Store the memory address of an object
- Also known as derived data types

- Java API contains lot of predefined reference data types
  - ex: Array, String

- Classes, interfaces, enums etc. defined by a programmer are called user defined data types

- Variables of reference data types are called reference variables
  - Reference variables can be assigned to null

PREDEFINED REFERENCE DATA TYPE

- String is a predefined class in Java API
- String is an object that represents a sequence of characters

```
String empName = new String("John");

String name = "Charlie";

String salutation = "Hello" + name;

String first = null;
```

USER DEFINED REFERENCE DATA TYPES

- Class is an example of user defined data type

```
class Employee {
  int empId;
}
```

- Objects of the Employee class can be created in the following way

```
Employee emp1 = new Employee();
```

- Employee is the user defined reference data type
- emp1 is the reference variable

CONCEPT

Operators

OPERATORS

- Java provides a wide range of operators for performing variety of operations
- Operators are classified based on the number of operands
  - Unary operators – have single operand
  - Binary operators – have two operands
  - Ternary operators – have three operands

CONCEPT

Unary operators

UNARY OPERATORS

- Work on single operand
- The following are unary operators
  - Increment and decrement : expr++, expr--, ++expr, --expr
  - Sign indicator : +expr, -expr
  - Bitwise Not : ~
  - Logical Not : !

INCREMENT AND DECREMENT OPERATORS

- The increment operator increases its operand by one.

```
x++;          // equivalent to x = x + 1
```

- The decrement operator decreases its operand by one.

```
x--;          // equivalent to x = x - 1
```

POSTFIX AND PREFIX

- postfix increment/decrement operator - follow the operand

```
int a = 10, b = 20, y = 0, z = 0;
y = a++;              // y = 10 and a becomes 11
z = b--;              // z = 20 and b becomes 19
```

- prefix increment/decrement operator - precede the operand

```
int a = 10, b = 20, y = 0, z = 0;
y = ++a;                    // y and a becomes 11
z = --b;                    // z and b becomes 19
```

manipalglobal
education services

SIGN INDICATORS

- The operator + and – are used to denote the sign

```
int a = +10;            // '+' is used to represent the value as
                           a positive number

int b = -2;             // '-' is used to represent the value as
                           a negative number
```

LOGICAL NOT

- Operates only on **boolean** operand.
- Negates the boolean value

```
boolean b1 = true;
boolean b2 = !b1            // b2 becomes false
```

CONCEPT

Binary operators

BINARY OPERATORS

Work on two operands on either side of the operator

- Arithmetic
- Relational
- Logical
- Shift and bitwise
- Assignment

ARITHMETIC OPERATORS

- Used in mathematical expressions
- The operands must be of a numeric type.
- **char** types can be used as it is a subset of int

Description	Operator
Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulus	%

```
int radius = 0;                    // Declare radius
double area = 0;                   // Declare area
radius = 20;                       // Assign a radius
area = radius * radius * 22 / 7;   // Compute area
```

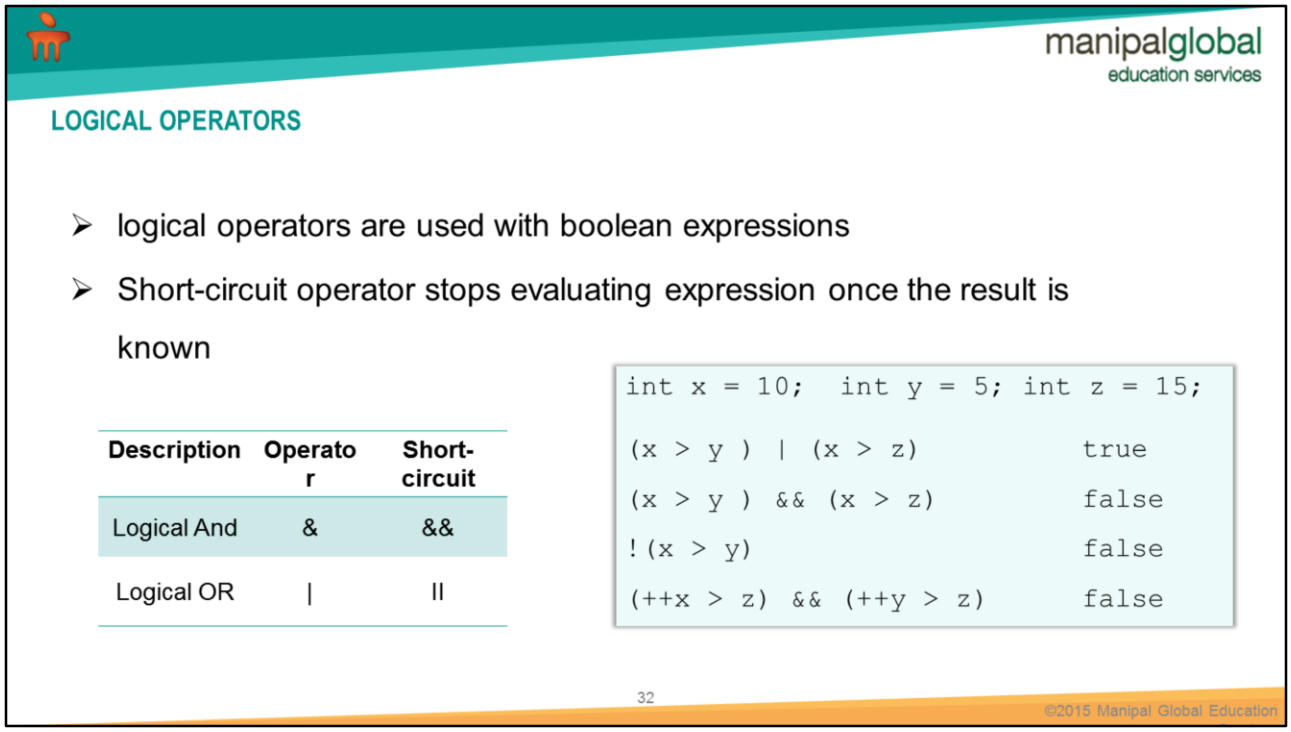
manipalglobal
education services

RELATIONAL OPERATORS

- Used to compare operands
- expression evaluates to true or false

```
int x = 10;
int y = 5;

(x == y)          false
(x > y)           true
```

Relational Operators	Description
>	tests if the left-hand value is greater than the right.
>=	tests if the left-hand value is greater than or equal to the right.
<	tests if the left-hand value is less than the right.
<=	tests if the left-hand value is less than or equal to the right.
==	tests equality and evaluates to true when two values are equal.
!=	tests inequality and evaluates to true if the two values are not equal.



The && and || operators perform Conditional-AND and Conditional-OR operations

They exhibit "short-circuiting" behavior,(i.e., the second operand is evaluated only if needed).

ASSIGNMENT AND COMPOUND ASSIGNMENT OPERATORS

Operator	Purpose
=	Assignment
+=	Addition assignment
–=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment

```
int x = 10; int y = 6;
x += 20;        | x = x + 20 -> 30
x -= 5          | x = x - 5  -> 5
x *= y/2        | x = x * (y/2) -> 30
```

CONCEPT

Ternary Operator

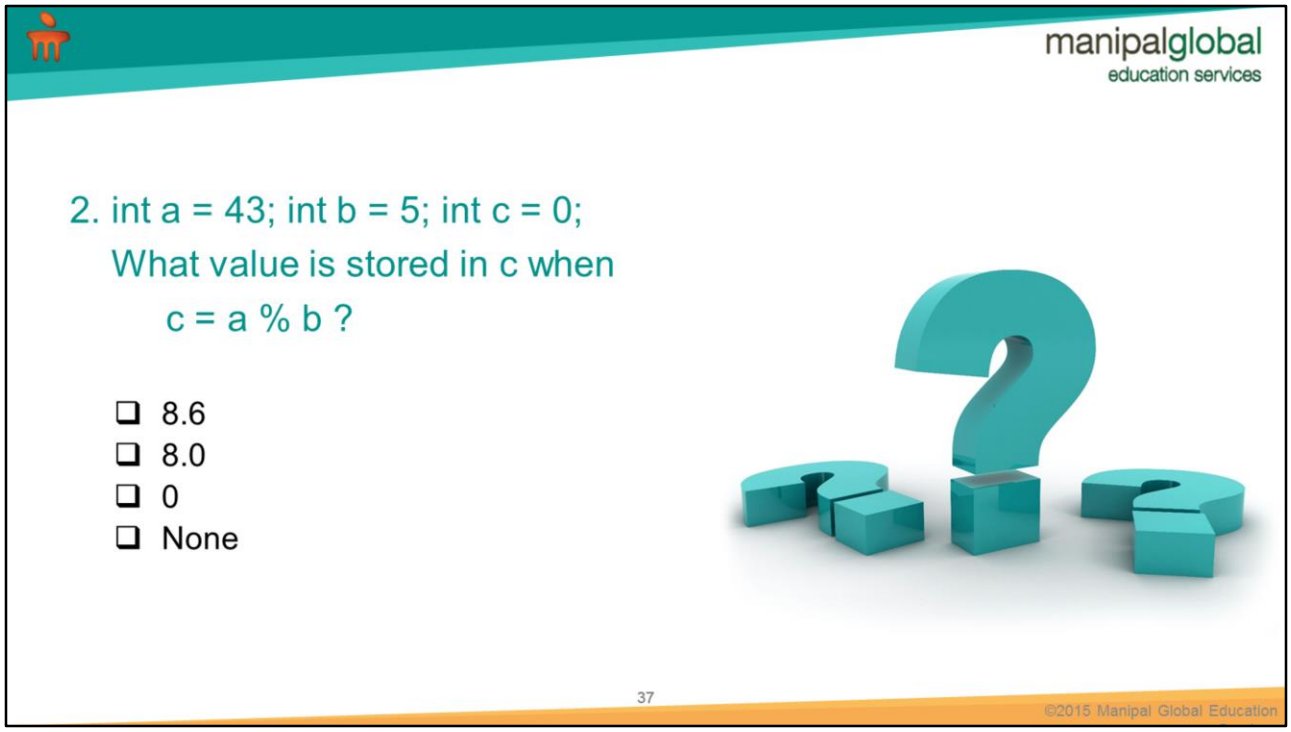
TERNARY OPERATOR

- Works on three operands
- Replaces if-else statement.
- The general form is *expression1* **?** *expression2* **:** *expression3*

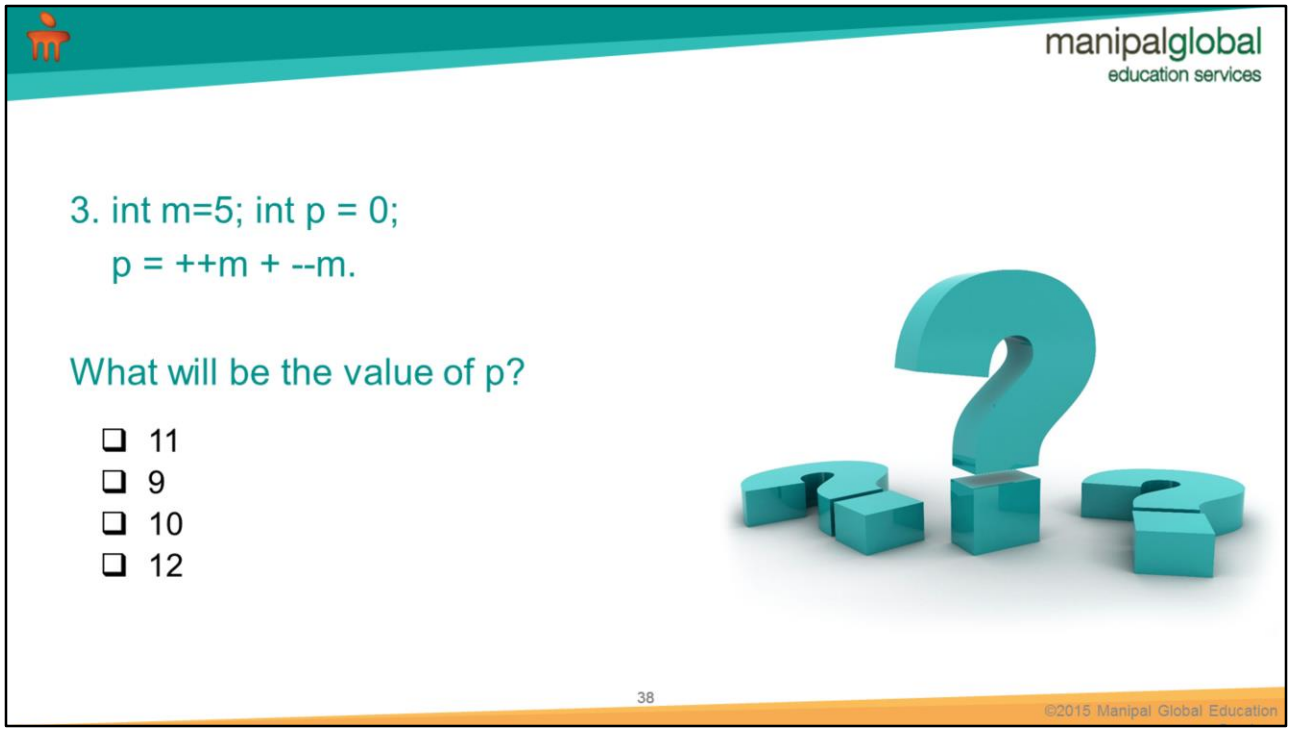
```
String result = ( a % 2 == 0 ) ? "Even" :"Odd";

equivalent of

        String result = null;
        if ( a % 2 == 0)
          result = "Even"
        else
           result = "Odd";
```


manipalglobal
education services
1. The statement n++ is equivalent to
++n
n+1
n=n+1
None
36
©2015 Manipal Global Education

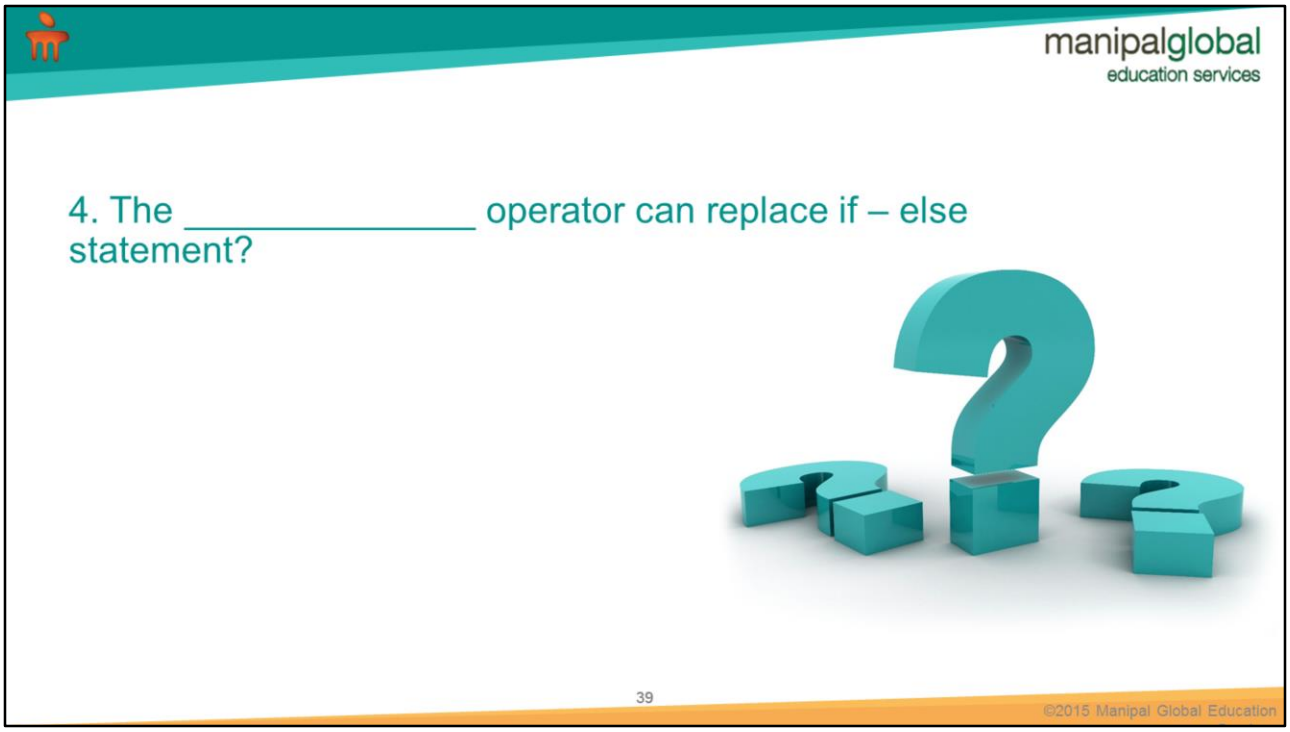
Answer: n=n+1


manipalglobal
education services
2. int a = 43; int b = 5; int c = 0;
What value is stored in c when
c = a % b ?
8.6
8.0
0
None
37
©2015 Manipal Global Education

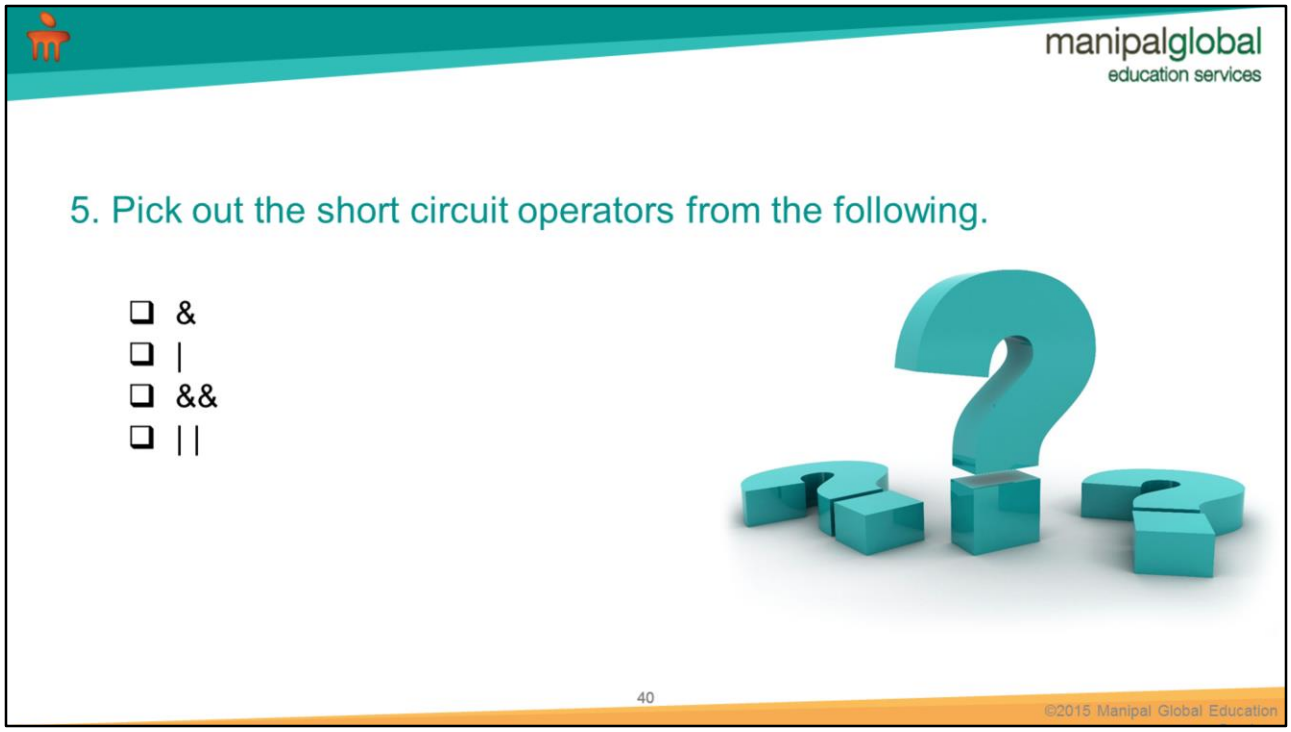
Answer: None


manipalglobal
education services
3. int m=5; int p = 0;
p = ++m + --m.
What will be the value of p?
11
9
10
12
38
©2015 Manipal Global Education

Answer: 11



Answer: ternary operator


manipalglobal
education services
5. Pick out the short circuit operators from the following.
&
|
&&
||
40
©2015 Manipal Global Education

Answer: && and | |

Refer following EduNxt Video for bitwise operators

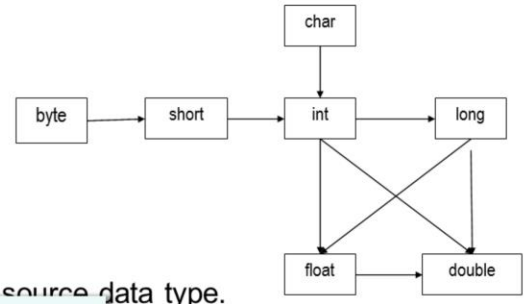
- M3L2L3_Demonstrating_use_of_bitwise_operators – Demo

CONCEPT

TYPE CASTING

TYPE CASTING

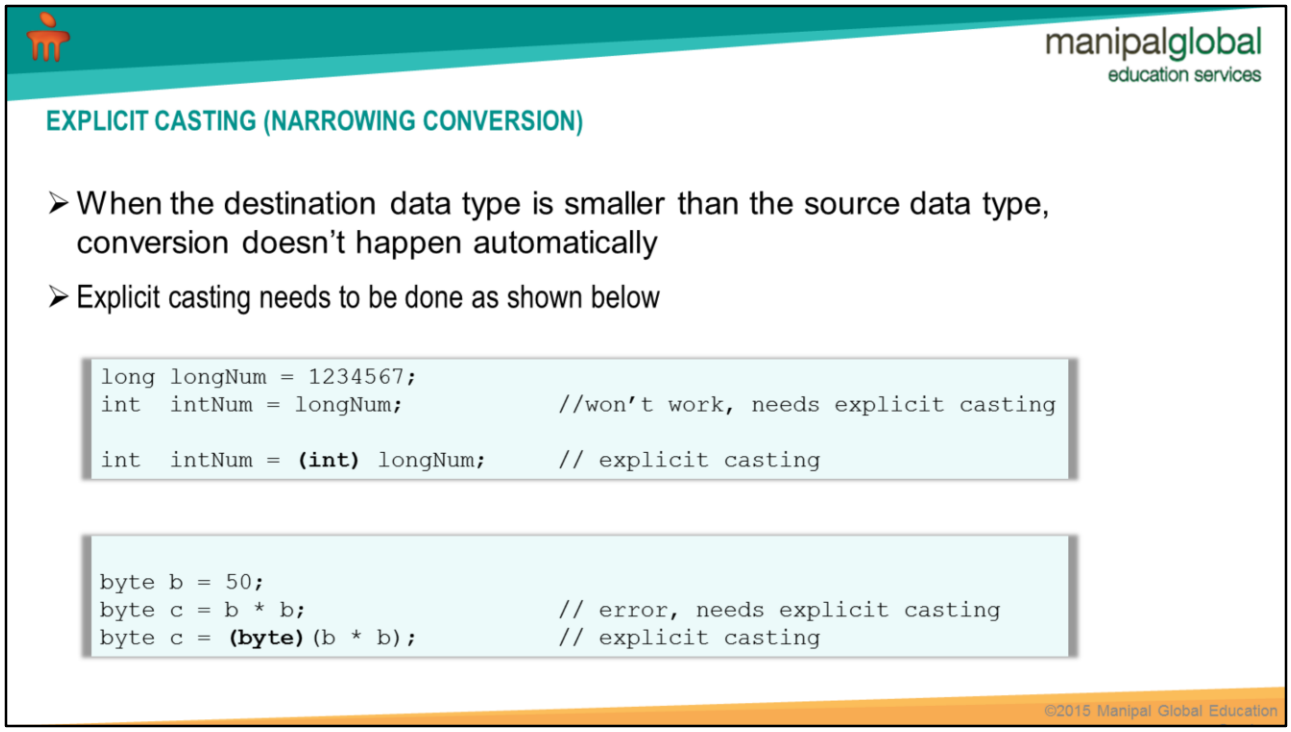
- Type Casting allows to convert primitive values from one data type to another
- Type Casting can be implicit or explicit



- Implicit casting (Widening conversion)
  - Conversion happens automatically
  - Takes place when
    - Two types are compatible
    - Destination data type is larger than the source data type.

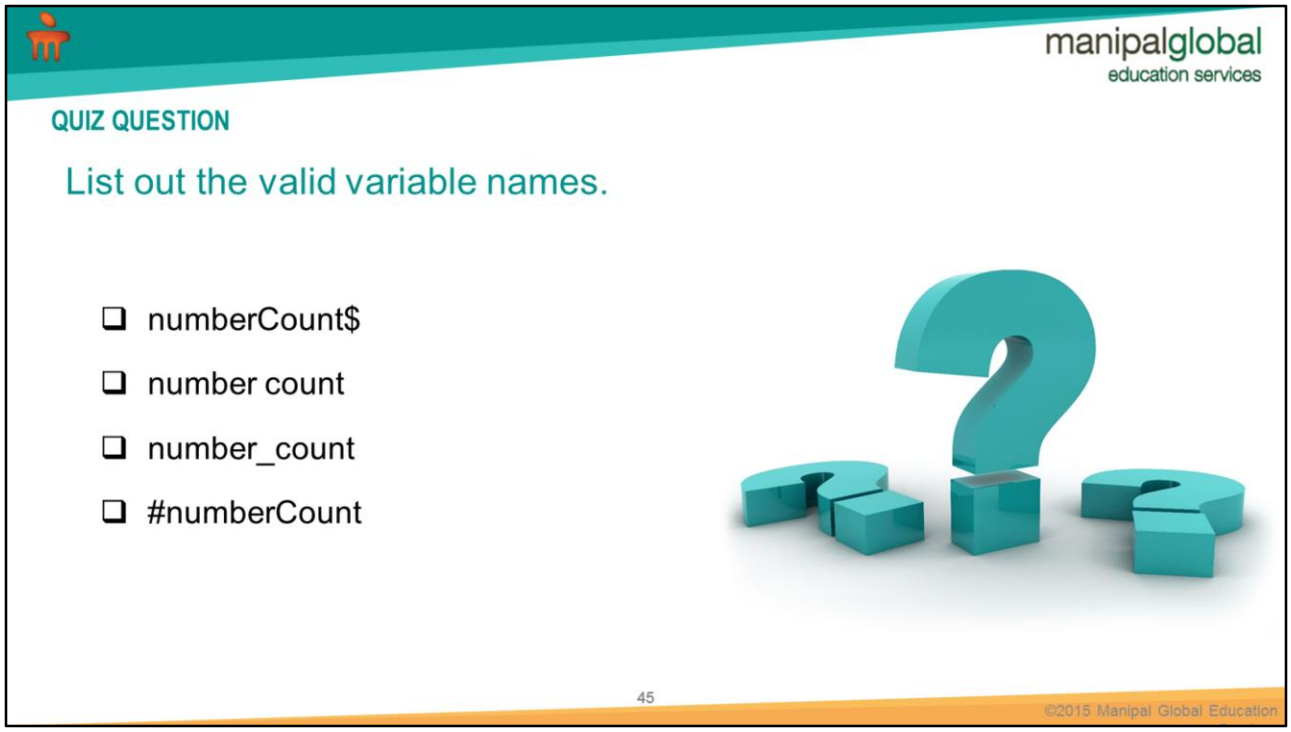
```
short s = 200;
int x = s;
```

```
float f = 100.25f;
double d = f;
```

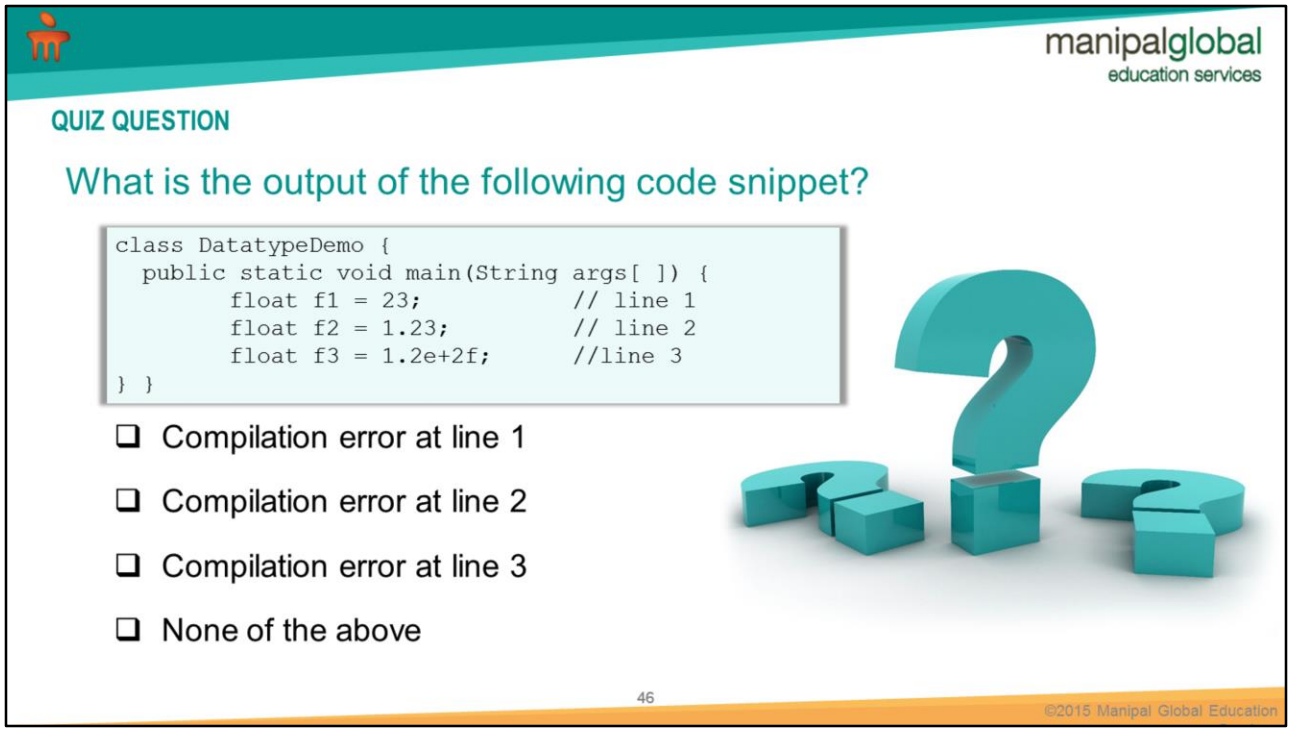


Explicit casting is done when the target type is larger than the type of the value in the right hand side

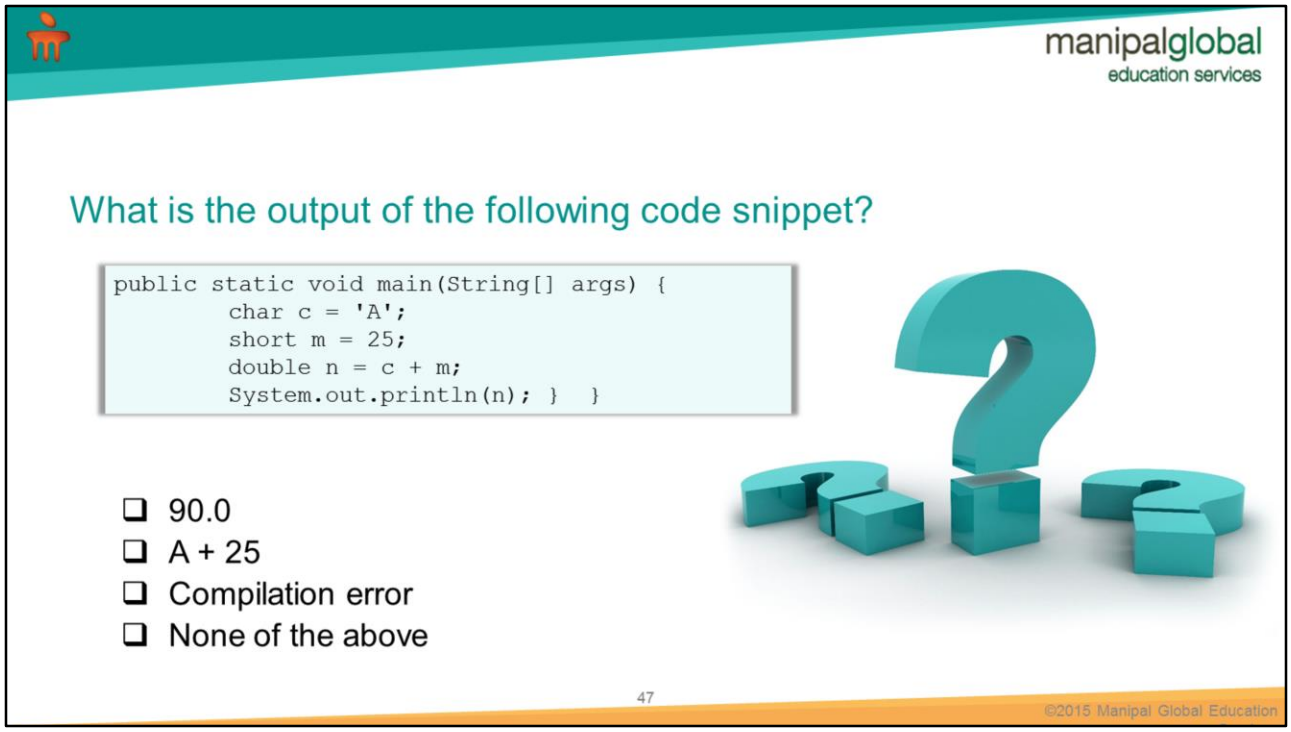
Conversion is referred to as narrowing



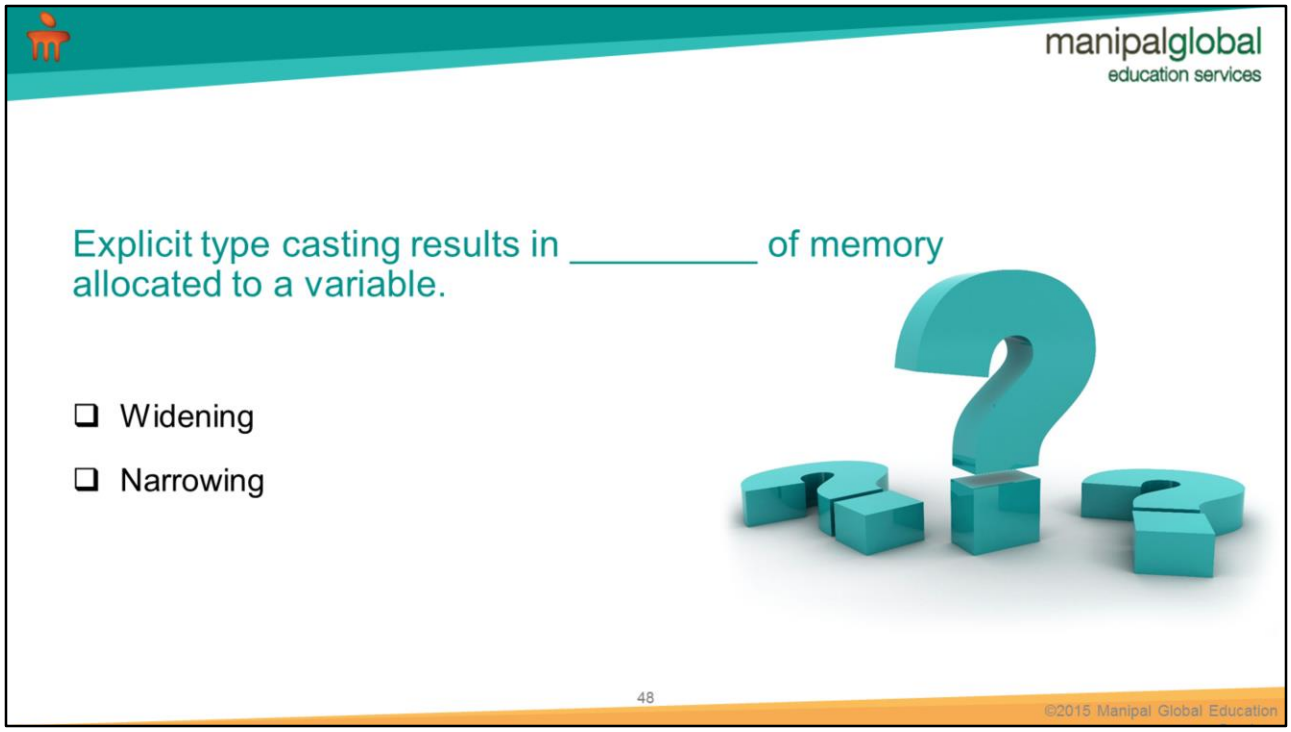
Answer: numberCount$, number_count



Answer: Compilation error at line 2


manipalglobal
education services
What is the output of the following code snippet?
public static void main(String[] args) {
char c = 'A';
short m = 25;
double n = c + m;
System.out.println(n); } }
90.0
A + 25
Compilation error
None of the above
47
©2015 Manipal Global Education

Answer: 90.0



Answer: Narrowing

REFERENCES

- Refer following videos on EduNxt
  - M1L2L1_Using_Eclipse_IDE – Demo
  - M3L1L4_Demonstration_of_using_variables_and_datatypes – Demo
  - M3L2L2_Demonstrating_use_of_basic_operators – Demo
  - M3L2L3_Demonstrating_use_of_bitwise_operators – Demo

SUMMARY

Variables, Data Types and Operators

SUMMARY

In this lesson, you've learned to:

- Identify and use Variables and Datatypes
- Classify and use operators
- Demonstrate Type Casting