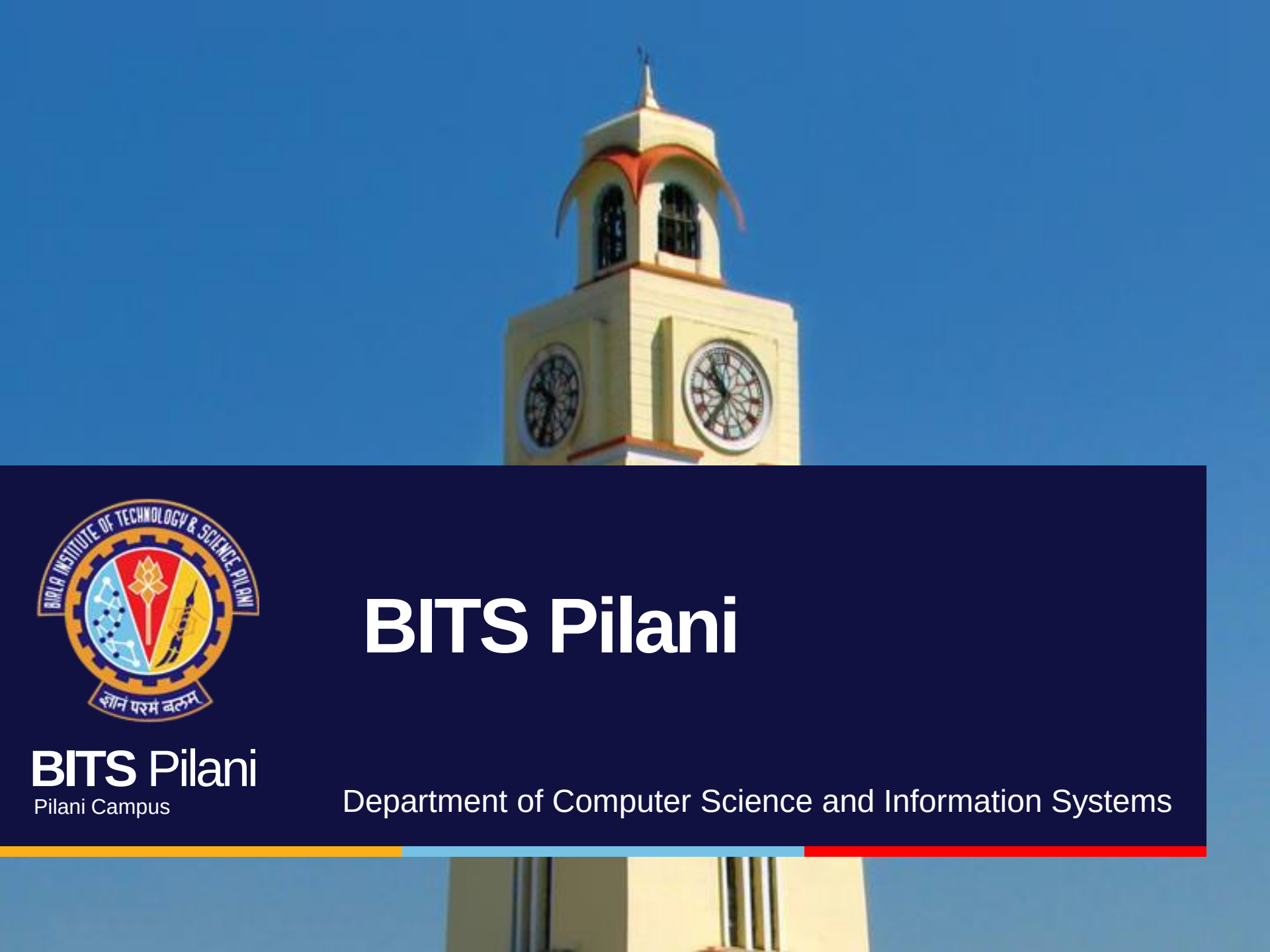


BITS Pilani

BITS Pilani
Pilani Campus

Department of Computer Science and Information Systems

Object Oriented Programming (CS F213)

Tanmaya Mahapatra
Tanmaya.mahapatra@pilani.bits-pilani.ac.in
Chamber# 6121-Y, NAB

Consultation Hour
By appointment through mail

The Bicycle

```
class Bicycle {
   int cadence = 0;
   int speed = 0;
   int gear = 1;

   void changeCadence(int newValue) {
        cadence = newValue;
   }
   void changeGear(int newValue) {
        gear = newValue;
   }
   void speedUp(int increment) {
        speed = speed + increment;
   }
   void applyBrakes(int decrement) {
        speed = speed - decrement;
   }
   void printStates() {
      System.out.println("cadence:" +cadence+ "speed:" +speed+ "gear:"
   +gear);
   }
} // End of Bicycle
```

Running Bicycle

```
class BicycleDemo {
        public static void main(String[] args) {

                // Create two different Bicycle objects
                Bicycle bike1 = new Bicycle();
                Bicycle bike2 = new Bicycle();

                // Invoke methods on those objects
                        bike1.changeCadence(50);
                bike1.speedUp(10);
                bike1.changeGear(2);
                bike1.printStates();

                bike2.changeCadence(50);
                bike2.speedUp(10);
                bike2.changeGear(2);
                bike2.changeCadence(40);
                bike2.speedUp(10);
                bike2.changeGear(3);
                bike2.printStates();
        }
}
```

Questions

1. When you compile a program written in the Java programming language, the compiler converts the human-readable source file into platform-independent code that a Java Virtual Machine can understand. What is this platform-independent code called?
Answer: Bytecode

2. Which of the following is *not* a valid comment:
a. / comment */**
b. /* comment */
c. /* comment
d. // comment
Answer: C

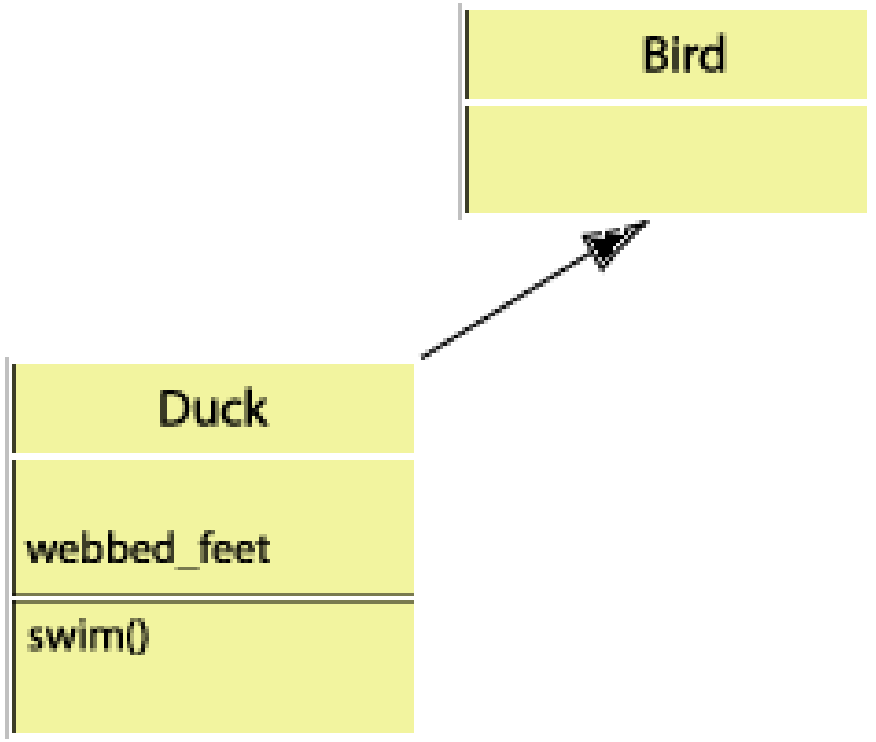
3. What is the correct signature of the main method?
Answer: public static void main(String[] args)

4. When declaring the main method, which modifier must come first, public or static?
Answer: They can be in either order, but the convention is public static

5. What parameters does the main method define?
Answer: The main method defines a single parameter, usually named args, whose type is an array of String objects

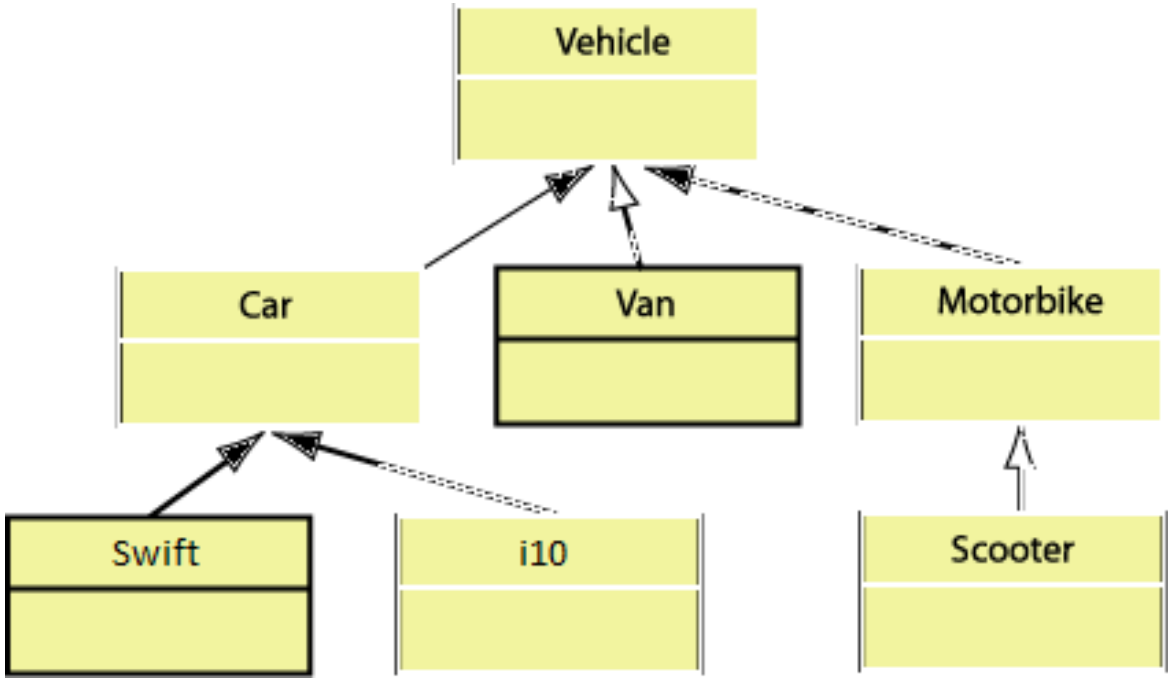
Inheritance

- If we have several descriptions with some commonality between these descriptions, we can group the descriptions and their commonality using inheritance to provide a compact representation of these descriptions.
- "What is a duck?“
  - "a bird that swims“
  - more accurately
  - "a bird that swims, with webbed feet.



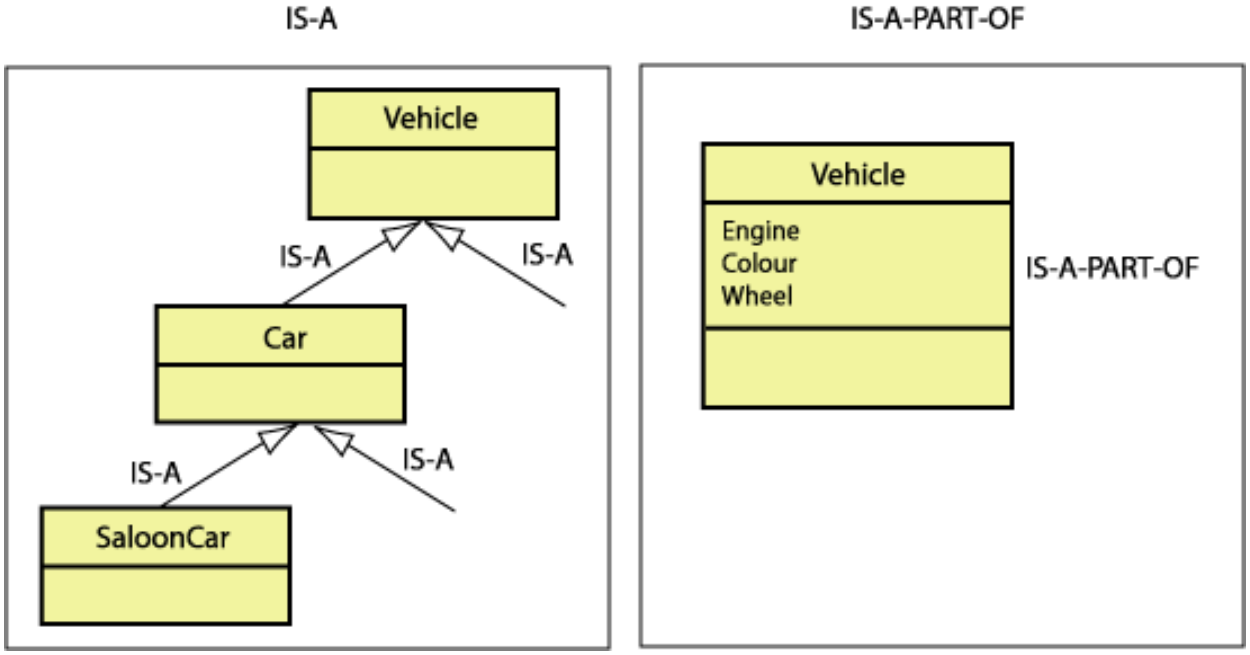
Inheritance – Another Example

- If were to be given an unstructured group of descriptions such as Car, Swift, i20, Van, Vehicle, Motorbike and Scooter, and asked to organize these descriptions by their differences.
- You might say that a Swift is a Car but has a long boot, whereas an i20 is a car with a small boot.



Inheritance

- One way to determine that you have organized your classes correctly is to check them using the "IS-A" and "IS-A-PART-OF" relationship checks.
- It is easy to confuse objects within a class and children of classes when you first begin programming with an OOP methodology.



Inheritance

- So, using inheritance the programmer can:
  - Inherit a behavior and add further specialized behavior
  - Inherit a behavior and replace it
  - Cut down on the amount of code that needs to be written and debugged

Polymorphism

- When a class inherits from another class it inherits both the states and methods of that class, so in the case of the Car class inheriting from the Vehicle class the Car class inherits the methods of the Vehicle class
  - **engineStart(), gearChange(), lightsOn()** etc.
- The Car class will also inherit the states of the Vehicle class
  - **isEngineOn, isLightsOn, numberWheels** etc.
- Polymorphism means "multiple forms".
- In OOP these multiple forms refer to multiple forms of the same method, where the exact same method name can be used in different classes, or the same method name can be used in the same class with slightly different parameters.
- There are two forms of polymorphism:
  - Over-riding
  - Over-loading

THANK YOU

innovate achieve lead

BITS Pilani

Pilani Campus

Object-Oriented Programming

Lecture -3 [Variables, Operators, Expressions, Statements and Blocks]

Lecture Outline

innovate achieve lead

Language Basics

- Variables
  - Primitive Data Types
- Operators
  - Assignment, Arithmetic, and Unary Operators
  - Equality, Relational, and Conditional Operators
  - Bitwise and Bit Shift Operators
- Expressions, Statements, and Blocks
- Control Flow Statements
  - **if – then** and **if – then – else**
  - **switch**
  - **while** and **do – while**
  - **for** and branching statements

Variables in Java

- We know that object stores its variables in its fields (state variables)
- What are the rules and conventions for naming a field?
- Besides int, what other data types are there?
- Do fields have to be initialized when they are declared?
- Are fields assigned a default value it they are not initialized?

In the Java programming language, the terms "field" and "variable" are both used; this is a common source of confusion among new developers, since both often seem to refer to the same thing.

Variables in Java

- The Java programming language defines the following kinds of variables
  - **Instance Variables (Non-Static Fields) -** objects store their individual states in "non-static fields", that is, fields declared without the static keyword. Non-static fields are also known as *instance variables* because their values are unique to each *instance* of a class. the currentSpeed of one bicycle is independent from the currentSpeed of another.
  - **Class Variables (Static Fields) -** A *class variable* is any field declared with the static modifier; this tells the compiler that there is exactly one copy of this variable in existence, regardless of how many times the class has been instantiated. A field defining the number of gears for a particular kind of bicycle could be marked as static since conceptually the same number of gears will apply to all instances. The code static int numGears = 6; would create such a static field. Additionally, the keyword final could be added to indicate that the number of gears will never change.

Variables in Java

- **Local Variables** - a method will often store its temporary state in local variables. There is no special keyword designating a variable as local; that determination comes entirely from the location in which the variable is declared — which is between the opening and closing braces of a method. As such, local variables are only visible to the methods in which they are declared; they are not accessible from the rest of the class.

- **Parameters** - parameters are always classified as "variables" not "fields".

Naming

- Variable names are case-sensitive. A variable's name can be any legal identifier — an unlimited-length sequence of Unicode letters and digits (a-z, A-Z, 0-9). Always begin your variable names with a letter. Subsequent characters may be letters, digits, dollar signs, or underscore characters.

- Spell one word in all lowercase letters.
  - Example: ***radius, area, weight etc***

- For more than one word, capitalize the first letter of each subsequent word.
  - Example: ***gearRatio, numberOfGears, monthlySalary etc***

- Capitalize every letter and separate subsequent words with the underscore character for variables that are made to store constant value, such that those with modifiers static, final, and static final both.
  - Example: ***static final NUM_GEARS = 6;***

Primitive Data Types

innovate achieve lead

Display 1.2 Primitive Types

TYPE NAME	KIND OF VALUE	MEMORY USED	SIZE RANGE
`boolean`	`true` or `false`	1 byte	not applicable
`char`	single character (Unicode)	2 bytes	all Unicode characters
`byte`	integer	1 byte	−128 to 127
`short`	integer	2 bytes	−32768 to 32767
`int`	integer	4 bytes	−2147483648 to 2147483647
`long`	integer	8 bytes	−9223372036854775808 to 9223372036854775807
`float`	floating-point number	4 bytes	$-3.40282347 \times 10^{+38}$ to $-1.40239846 \times 10^{-45}$
`double`	floating-point number	8 bytes	$\pm 1.76769313486231570 \times 10^{+308}$ to $\pm 4.94065645841246544 \times 10^{-324}$

Size of Primitive Types

- Because Java byte-code runs on the Java Virtual Machine, the size (number of bytes) for each primitive type is fixed.
- The size is not dependent on the actual machine/device on which the code executes.
- There is no `sizeof` operator in Java as there is in C – it's not necessary.

Primitive Data Types (Contd.)

- Java programming language provides special support for character strings via the java.lang.String class.
- Enclosing your character string within double quotes will automatically create a new String object;
- for example, String s = “this is a string”;
- String objects are *immutable*, which means that once created, their values cannot be changed.
- The String class is not technically a primitive data type, but considering the special support given to it by the language, you'll probably tend to think of it as such.

Default values

- It's not always necessary to assign a value when a field is declared. Fields that are declared but not initialized will be set to a reasonable default by the compiler. Generally speaking, this default will be zero or null, depending on the data type. Relying on such default values, however, is generally considered bad programming style.

- **Data Type Default Value (for fields)**
  - byte - 0
  - short - 0
  - int - 0
  - long - 0L
  - float - 0.0f
  - double - 0.0d
  - char - '\u0000'
  - String (or any object) - null
  - Boolean - false

The compiler never assigns a default value to an uninitialized local variable. If you cannot initialize your local variable where it is declared, make sure to assign it a value before you attempt to use it. Accessing an uninitialized local variable will result in a compile-time error

Questions

- The term "instance variable" is another name for
  - **non-static field**
- The term "class variable" is another name for
  - **static field**
- A local variable stores temporary state; it is declared inside a
  - **method**
- A variable declared within the opening and closing parenthesis of a method is called a
  - **parameter**
- Character strings are represented by the class
  - **java.lang.String**

Operators

- Assignment, Arithmetic, and Unary Operators
- Equality, Relational, and Conditional Operators
- Bitwise and Bit Shift Operators

Operator Precedence

Operators	Precedence
postfix	*expr*++ *expr*–
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Operators with higher precedence are evaluated before operators with relatively lower precedence. Operators on the same line have equal precedence. When operators of equal precedence appear in the same expression, a rule must govern which is evaluated first. All binary operators except for the assignment operators are evaluated from left to right; assignment operators are evaluated right to left.

Arithmetic Operators & Expressions

- If an arithmetic operator is combined with **`int`** operands, then the resulting type is **`int`**
- If an arithmetic operator is combined with one or two **`double`** operands, then the resulting type is **`double`**
- If different types are combined in an expression, then the resulting type is the right-most type on the following list that is found within the expression

  `byte`→**`short`**→**`int`**→**`long`**→**`float`**→**`double`**

  `char`→**`int`**

  – Exception: If the type produced should be **`byte`** or **`short`** (according to the rules above), then the type produced will actually be an **`int`**

Integer and Floating point division

- When one or both operands are a floating-point type, division results in a floating-point type

  `15.0/2` evaluates to `7.5`

- When both operands are integer types, division results in an integer type
  - Any fractional part is discarded
  - The number is not rounded

    `15/2` evaluates to `7`

- Be careful to make at least one of the operands a floating-point type if the fractional portion is needed

Type Casting

- A *type cast* takes a value of one type and produces a value of another type with an "equivalent" value
  - If `n` and `m` are integers to be divided, and the fractional portion of the result must be preserved, at least one of the two must be type cast to a floating-point type **before** the division operation is performed

```
double ans = n / (double)m;
```

  - Note that the desired type is placed inside parentheses immediately in front of the variable to be cast
  - Note also that the type and value of the variable to be cast does not change

More on Type Casting

- When type casting from a floating-point to an integer type, the number is truncated, not rounded

  `(int)2.9` evaluates to `2`, not `3`

- When the value of an integer type is assigned to a variable of a floating-point type, Java performs an automatic type cast called a *type conversion*

```
double d = 5;
```

- In contrast, it is illegal to place a `double` value into an `int` variable without an explicit type cast

```
int i = 5.5;        // Illegal
int i = (int)5.5    // Correct
```

Questions

- Consider the following code snippet.

```
int i = 10;
int n = i++%5;
```

  - What are the values of i and n after the code is executed?
    - Answer: i is 11, and n is 0
  - What are the final values of i and n if instead of using the postfix increment operator (i++), you use the prefix version (++i))?
    - Answer: i is 11, and n is 1
- To invert the value of a boolean, which operator would you use?
  - Answer: The logical complement operator "!"
- Which operator is used to compare two values, = or == ?
  - Answer: The == operator is used for comparison, and = is used for assignment
- Explain the following code sample:

```
result = someCondition ? value1 : value2
```

  - Answer: This code should be read as: "If someCondition is true, assign the value of value1 to result. Otherwise, assign the value of value2 to result."

Control Flow Statements

- The if-then and if-then-else Statements
- The switch Statement
- The while and do-while Statements
- The for Statement
- Branching Statements

THANK YOU