Literals and Ranges of All Primitive Data Types

- All six number types in Java are signed, so they can be positive or negative.
- Use the formula -2(bits-1) to 2(bits-1)-1 to determine the range of an integer type.
- A char is really a 16-bit unsigned integer.
- Literals are source code representations of primitive data types, or String.
- Integers can be represented in octal (0127), decimal (1245), and hexadecimal (0XCAFE).
- Numeric literals cannot contain a comma.
- A char literal can be represented as a single character in single quotes (‘A’).
- A char literal can also be represented as a Unicode value (‘\u0041’).
- A char literal can also be represented as an integer, as long as the integer is less than 65536.
- A boolean literal can be either true or false.
- Floating-point literals are always double by default; if you want a float, you must append an *F* or *f* to the literal.

Array Declaration, Construction, and Initialization

- Arrays can hold primitives or objects, but the array itself is always an object.
- When you declare an array, the brackets can be to the left or right of the variable name.
- It is never legal to include the size of an array in the declaration.

- You must include the size of an array when you construct it (using new) unless you are creating an anonymous array.
-
- Elements in an array of objects are not automatically created, although primitive array elements are given default values.
-
- You'll get a NullPointerException if you try to use an array element in an object array, if that element does not refer to a real object.

- Arrays are indexed beginning with zero. In an array with three elements, you can access element 0, element 1, and element 2.

- You'll get an ArrayIndexOutOfBoundsException if you try to access outside the range of an array.

- Arrays have a length variable that contains the number of elements in the array.

- The last index you can access is always one less than the length of the array.

- Multidimensional arrays are just arrays of arrays.

- The dimensions in a multidimensional array can have different lengths.

- An array of primitives can accept any value that can be promoted implicitly to the declared type of the array. For example, a byte variable can be placed in an int array.

- An array of objects can hold any object that passes the IS-A (or instanceof) test for the declared type of the array. For example, if Horse extends Animal, then a Horse object can go into an Animal array.

- If you assign an array to a previously declared array reference, the array you're assigning must be the same dimension as the reference you're assigning it to.

- You can assign an array of one type to a previously declared array reference of one of its supertypes. For example, a Honda array can be assigned to an array declared as type Car (assuming Honda extends Car).

Using a Variable or Array Element That Is Uninitialized and Unassigned

- When an array of objects is instantiated, objects within the array are not instantiated automatically, but all the references get the default value of null.
- When an array of primitives is instantiated, all elements get their default values.
- Just as with array elements, instance variables are always initialized with a default value.
- Local/automatic/method variables are never given a default value. If you attempt to use one before initializing it, you'll get a compiler error.

Command-Line Arguments to Main

- Command-line arguments are passed to the String array parameter in the main method.
- The first command-line argument is the first element in the main String array parameter.
- If no arguments are passed to main, the length of the main String array parameter will be zero.

Properties of main()

- It must be marked static.
- It must have a void return type.
- It must have a single String array argument; the name of the argument is flexible, but the convention is args.
- For the purposes of the exam, assume that the main() method must be public.
- Improper main() method declarations (or the lack of a main() method) cause a runtime error, not a compiler error.

- In the declaration of main(), the order of public and static can be switched, and args can be renamed.

- Other overloaded methods named main() can exist legally in the class, but if none of them match the expected signature for the main() method, then the JVM won't be able to use that class to start your application running.

Java Operators

- The result of performing most operations is either a boolean or a numeric value.
- Variables are just bit holders with a designated type.
- A reference variable's bits represent a way to get to an object.
- An unassigned reference variable's bits represent null.
- There are 12 assignment operators: =, *=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=, |=.
- Numeric expressions always result in at least an int-sized result—never smaller.
- Floating-point numbers are implicitly doubles (64 bits).
- Narrowing a primitive truncates the high-order bits.
- Two's complement means: flip all the bits, then add 1.
- Compound assignments (e.g. +=) perform an automatic cast.

Reference Variables

- When creating a new object, e.g., Button b = new Button();, three things happen:
    - Make a reference variable named b, of type Button
    - Create a new Button object
    - Refer the reference variable b to the Button object

- Reference variables can refer to subclasses of the declared type but not super classes.

String Objects and References

- String objects are immutable, cannot be changed.

- When you use a String reference variable to modify a String:
    - A new string is created (the old string is immutable).
    - The reference variable refers to the new string.

Comparison Operators

- Comparison operators always result in a boolean value (true or false).

- There are four comparison operators: >, >=, <, <=.

- When comparing characters, Java uses the ASCII or Unicode value of the number as the numerical value.

instanceof Operator

- instanceof is for reference variables only, and checks for whether this object is of a particular type.

- The instanceof operator can be used only to test objects (or null) against class types that are in the same class hierarchy.

- For interfaces, an object is “of a type” if any of its superclasses implement the interface in question.

Equality Operators

- Four types of things can be tested: numbers, characters, booleans, reference variables.

- There are two equality operators: == and !=.

Arithmetic Operators

- There are four primary operators: add, subtract, multiply, and divide.
- The remainder operator returns the remainder of a division.
- When floating-point numbers are divided by zero, they return positive or negative infinity.
- When the remainder operator performs a floating-point divide by zero, it will not cause a runtime exception.
- When integers are divided by zero, a runtime ArithmeticException is thrown.
- When the remainder operator performs an integer divide by zero, a runtime ArithmeticException is thrown.

String Concatenation Operator

- If either operand is a String, the + operator concatenates the operands.
- If both operands are numeric, the + operator adds the operands.

Increment/Decrement Operators

- Prefix operator runs before the value is used in the expression.
- Postfix operator runs after the value is used in the expression.
- In any expression, both operands are fully evaluated before the operator is applied.
- Final variables cannot be incremented or decremented.

Shift Operators

- There are three shift operators: >>, <<, >>>; the first two are signed, the last is unsigned.
- Shift operators can only be used on integer types.
- Shift operators can work on all bases of integers (octal, decimal, or hexadecimal).

- Bits are filled as follows:
  - << fills the right bits with zeros.
  - >> fills the left bits with whatever value the original sign bit (leftmost bit) held.
  - >> fills the left bits with zeros (negative numbers will become positive).

- All bit shift operands are promoted to at least an int.

- For int shifts > 32 or long shifts > 64, the actual shift value is the remainder of the right operand / divided by 32 or 64, respectively.

Bitwise Operators

- There are three bitwise operators—&, ^, |—and a bitwise complement, operator ~.

- The & operator sets a bit to 1 if both operand's bits are set to 1.

- The ^ operator sets a bit to 1 if exactly one operand's bit is set to 1.

- The | operator sets a bit to 1 if at least one operand's bit is set to 1.

- The ~ operator reverses the value of every bit in the single operand.

Ternary (Conditional Operator)

- Returns one of two values based on whether a boolean expression is true or false.

- The value after the ? is the 'if true return'.

- The value after the : is the 'if false return'.

Casting

- Implicit casting (you write no code) happens when a widening conversion occurs.

- Explicit casting (you write the cast) happens when a narrowing conversion occurs.

- Casting a floating point to an integer type causes all digits to the right of the decimal point to be lost (truncated).
- Narrowing conversions can cause loss of data—the most significant bits (leftmost) can be lost.

Logical Operators

- There are four logical operators: &, |, &&, ||.
- Logical operators work with two expressions that must resolve to Boolean values.
- The && and & operators return true only if both operands are true.
- The || and | operators return true if either or both operands are true.
- The && and || operators are known as short-circuit operators.
- The && operator does not evaluate the right operand if the left operand is false.
- The || does not evaluate the right operand if the left operand is true.
- The & and | operators always evaluate both operands.

Passing Variables into Methods

- Methods can take primitives and/or object references as arguments.
- Method arguments are always copies—of either primitive variables or reference variables.
- Method arguments are never actual objects (they can be references to objects).
- In practice, a primitive argument is a completely detached copy of the original primitive.
- In practice, a reference argument is another copy of a reference to the original object.