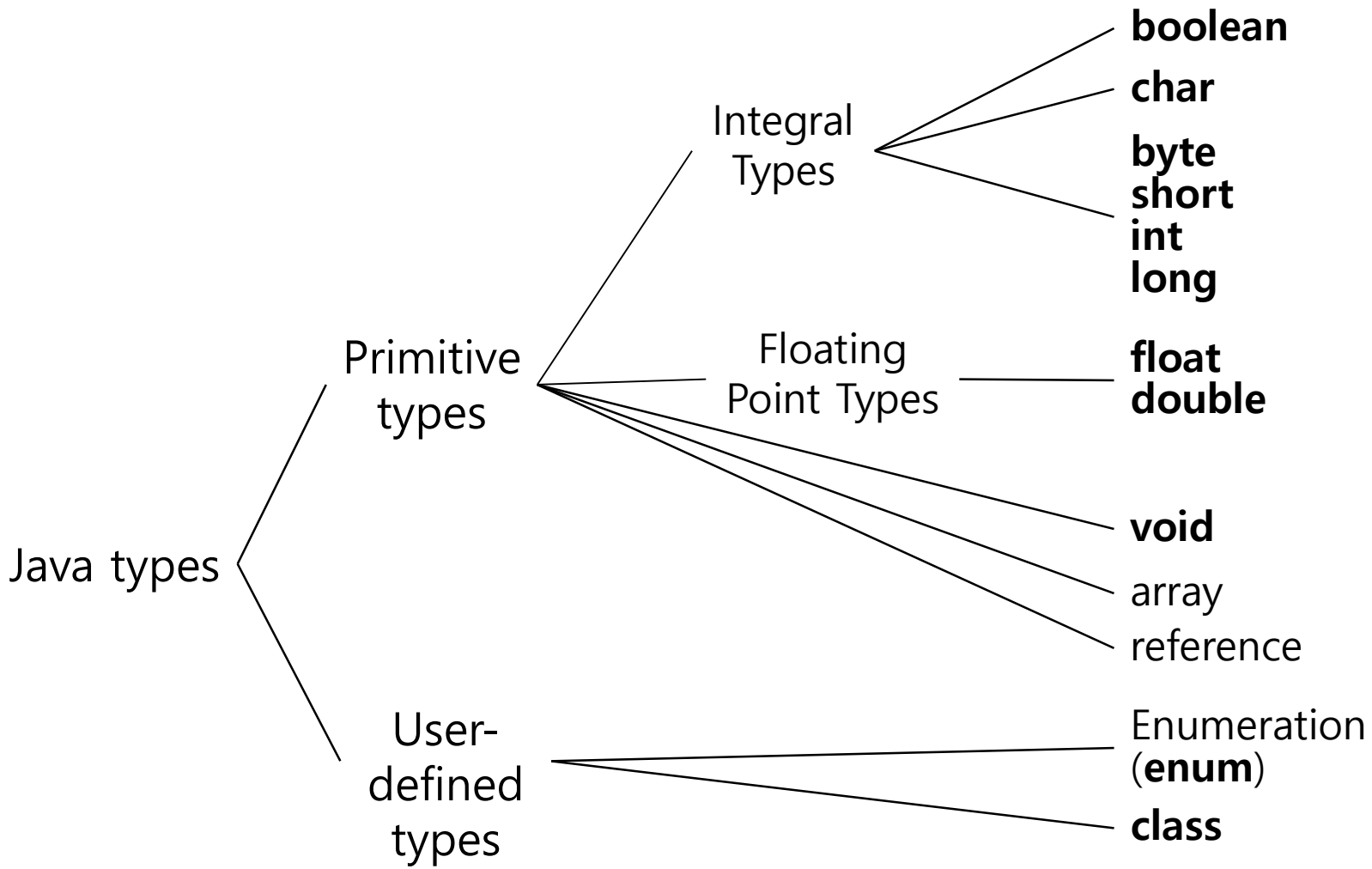
Java Fundamentals

- ❖Types
- ❖Operators
- ❖Wrapper Classes

- ❖String
- ❖Array
- ❖Enumeration
- ❖Constants

- ❖Inputs and Outputs
- ❖Date & Time

Java Types: Overview



Operators in Java

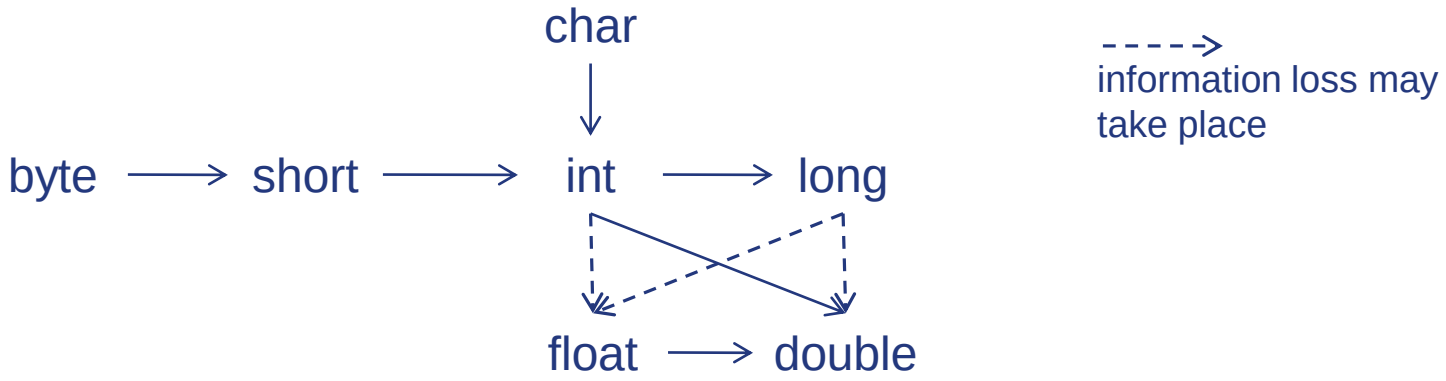
Categories	Operators
Assignment	=, +=, -=, *=, /=
Arithmetic	+, -, *, /, %, ++,, --
Equality and Relational Operators	==, !=, >, >=, <, <=
Conditional	&& ||, ? : (= if then else)
Type Comparison	instanceof
Bitwise and Bit Shift	<<, >>, >>>, &, ^, |, ~

Primitive Types

Category	Type	Bytes	Range(inclusive)	Literals
Integer	int	4	-2^{31} (=-2,147,483,648) to 2^{31}-1 (=2,147,483,647)	26 **0**32 **0x**1a
	short	2	-2^{15} (=-32768) to 2^{15}-1 (=32767)	
	long	8	-2^{63} to 2^{63}-1	26**L**
	byte	1	-2^{7}(=-128) to 2^{7}-1(=127)	
Floating-Point	float	4	approximately 3.403E+38 (6-7 significant decimal digits)	123.4**f** 123.4**F**
	double	8	approximately 1.198E+308 (15 significant decimal digits)	123.4
Character	char	2	Any character supported by Unicode	'A' '한' '₩u203B'
Truth value	boolean	1(?)		true, false

Type Conversion

- Impicit conversion: legal conversions between numeric types



- Explicit cast

```
double x = 9.997 ;
int nx1 = (int) x ;
Int nx2 = (int) Math.round(x) ;
```

Unicode

- Unicode is an encoding scheme for representing various characters including Alphabet, Chinese, Koreans, and Japanese.
- Java supports Unicode. That is, we can use all the characters supported by the Unicode.

```
public class Unicode {
    public static void main(String[] args) {
        // Korean
        System.out.print("안녕하세요! ") ;
        char[] korean = {'\uC790', '\uBC14'} ;
        System.out.println(korean) ;

        // Japanese
        char[] japanese = {'\u3051', '\u304F'} ;
        System.out.println(japanese) ;

        // Symbols
        char[] symbol = {'\u2020', '\u203B'} ;
        System.out.println(symbol) ;
    }
}
```

```
안녕하세요! 자바
け く
†※
```

- For the Unicode, visit http://unicode.org/charts/

Beyond Basic Arithmetic: Math Class

- The Math class provides methods and constants for doing more advanced mathematical computation.

```
public class MathExample {
   public static void main(String[] args) {
      System.out.println(Math.abs(-10)) ;
      System.out.println(Math.PI) ;
   }
}
```

Categories	methods
Basic math methods	abs, ceil, floor, round, min, max
Exponential and Logarithmic Methods	exp,log, pow, sqrt
Trigonometric methods	sin, cos, tan, asin, acos, atan
Random numbers (0.0 – 1.0)	random

WRAPPER CLASSES

Wrapper Classes

- Java supports wrapper classes for primitive numeric types
  - **int** intValue = 10;
  - **Integer** integerValue = intValue;

Primitive types	Wrapper Classes
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Conversion: Primitive Type → Wrapper

```
public class Conversion2Wrapper {
    public static void main(String[] args) {
        // 1. using constructor
        // The constructor Integer(int) is deprecated since version 9
        Integer integer1 = new Integer(10);
        System.out.println(integer1);

        // 2. using static factory method: valueOf()
        // The static factory valueOf(int) is generally a better choice,
        // as it is likely to yield significantly better space and time performance
        Integer integer2 = Integer.valueOf(20);
        System.out.println(integer2);

        System.out.println(integer1 + integer2);
    }
}
```

```
10
20
30
```

Conversion: Wrapper → Primitive Type

- Use xxxValue()

```
public class Conversion2PrimitiveType {

    public static void main(String[] args) {
        Integer integer1 = new Integer(10);
        int intValue = integer1.intValue();

        Integer integer2 = Integer.valueOf(118);
        short shortIntValue = integer2.shortValue();

        Integer integer3 = integer1 + integer2;
        long longIntValue = integer3.longValue();

        byte byteValue = integer3.byteValue(); // overflow
        System.out.println(byteValue); // -128, not 128
    }
}
```

Wrapper Classes: Autoboxing and Unboxing

- **Boxing**
  - Automatic conversion: primitive type ➔ wrapper class
  - Character characterValue = 'A';
  - Integer integerValue = 10;

- **Unboxing**
  - Automatic conversion: wrapper class ➔ primitive type
  - int intValue1 = integerValue;
  - Char charValue = ch;

Auto Boxing

```
import java.util.ArrayList;
import java.util.List;

public class AutoBoxing {
    public static void main(String[] args) {
        List<Integer> integerList = new ArrayList<>();
        for (int i = 1; i < 10; i ++) {
            integerList.add(i);     // int to Integer; add(Integer.valueOf(i))
        }
        System.out.println(integerList); // [1, 2, 3, 4, 5, 6, 7, 8, 9]
    }
}
```

Auto Unboxing

```
public class AutoUnboxing {
    public static void main(String[] args) {
        List<Integer> integerList = new ArrayList<>();
        for ( int i = 1; i <= 5; i ++ )
          integerList.add(i); // auto boxing
       System.out.println(integerList);                // [1, 2, 3, 4, 5]

       int sumOfEven = 0;
       for ( Integer i: integerList ) {
            if ( i % 2 == 0 )                          // Integer to int
              sumOfEven += i ;                         // Integer to int
        }
       System.out.println(sumOfEven);                  // 6
    }
}
```

Wrapper Classes: Useful Features

- Wrapper classes provide useful variables.
  - MIN_VALUE, MAX_VALUE, SIZE for integer types
  - NEGATIVE_INFINITY, POSITIVE_INFINITY for floating-point types

```
…
byte b ;
if ( integerValue <= Byte.MAX_VALUE )
    b = integerValue.byteValue() ;
else
    b = 0 ;

Double d ;
if ( Double.isInfinite(d) ) …
if ( d.isInfinite() ) …

if ( Double.isNaN(d) ) …
if ( d.isNaN()) …
```

When to Use Wrapper Classes

- **Collections** in Java deal only with objects; to store a primitive type in one of these classes, you need to wrap the primitive type in a class.

  - `List<int> ints = new ArrayList<>();` // X
  - `List<Integer> integers = new ArrayList<>();` // O

STRING

String

```
public class StringExample {
   public static void main(String[] args) {
      String greeting = "Hello" ;

      // length, charAt
      for ( int i = 0 ; i < greeting.length() ; i ++ )
         System.out.println(greeting.charAt(i)) ;

      // substring
      String hel = greeting.substring(0, 3) ;
      System.out.println(hel);    // Hel

      // concatenation
      String language = "Java !" ;
      String msg = greeting + " " + language ;
      System.out.println("Welcome to " + msg) ;    // Welcome to Hello Java !

      // equality, use equals; DO NOT USE ==
      if ( greeting.equals("hello"))
         System.out.println("Exactly same!") ;
      if ( greeting.equalsIgnoreCase("hello"))
         System.out.println("Same when case ignored") ;       // this executed
```

```
        // comparison
        if ( greeting.compareTo(language) < 0 )
            System.out.println(greeting + " comes before " + language) ;  // this executed
        else if ( greeting.compareTo(language) > 0 )
            System.out.println(greeting + " comes after " + language) ;
        else
            System.out.println(greeting + " equals with " + language) ;

        // replacement
        String greeting2 = greeting.replace('l', 'L') ;
        System.out.println("The original string: " + greeting + " After replacement: " + greeting2) ;

        // indexOf, lastIndexOf
        System.out.println(greeting.indexOf('l')) ;       // 2
        System.out.println(greeting.lastIndexOf('l')) ; // 3
        System.out.println(greeting.indexOf('L')) ;       // -1
        System.out.println(greeting.indexOf("lo")) ;    // 3

        // startsWith, endsWith
        System.out.println(greeting.startsWith("He"));              // true
        System.out.println(greeting.startsWith("he"));              // false
        System.out.println(greeting.endsWith("lo"));                // true
        System.out.println(greeting.startsWith("hlo"));             // false

        System.out.println(String.join("-", "I", "Love", "Java"));    // I-Love-Java
    }
}
```

Splitting String

```
public class StringSplitExample {
   public static void main(String[] args) {
      String message1 = "Hello₩tWorld₩t₩tI₩t₩tLove₩tJava";

      String[] words11 = message1.split("₩t");
      for ( int i=0; i < words11.length; i++ )
         System.out.println(i + ": [" + words11[i] + "]");

      String[] words12 = message1.split("₩t+");
      for ( int i=0; i < words12.length; i++ )
         System.out.println(i + ": [" + words12[i] + "]");

      String message2 = "Hello₩tWorld₩nLove  Java";

      String[] words21 = message2.split("₩₩s");
      for ( int i=0; i < words21.length; i++ )
         System.out.println(i + ": [" + words21[i] + "]");

      String[] words22 = message2.split("₩₩s+");
      for ( int i=0; i < words22.length; i++ )
         System.out.println(i + ": [" + words22[i] + "]");
   }
}
```

Conversion between Number and String

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

      // String ==> Number
      String intString = "100", floatString = "1.234F" ;
      int a = Integer.valueOf(intString); // Auto unboxing: Integer -> int
      float b = Float.valueOf(floatString);          // Auto unboxing: Float -> float
      System.out.println( a + " " + b ) ;  // 100 1.234

      // or use parseXXX()
      a = Integer.parseInt(intString) ;
      b = Float.parseFloat(floatString) ;

      // Number ==> String
      Integer intValue = 100 ;
      String strI = intValue.toString() ;
      System.out.println(strI) ;  // 100

      float f = 1.234F ;
      String strF = Float.valueOf(f).toString() ;
      System.out.println(strF) ; // 1.234
   }
}
```

Formatting String

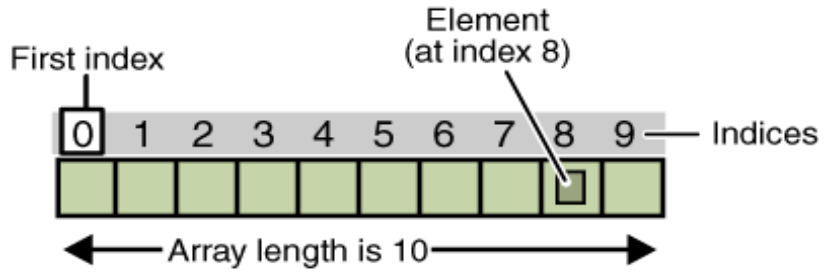
```
public class StringFormat {
   public static void main(String[] args) {
      String str1 = String.format("%d", 101);                    // Integer value
      String str2 = String.format("|%15d|", 101);                // length and right-justified
        String str3 = String.format("|%-15s|", "Hello, Java");   // left-justified
        String str4 = String.format("|%015f|", 101.00);          // leading zeros
        String str5 = String.format("|%15.2f|", 101.00);

        String str6 = String.format("%x", 101);                  // Hexadecimal value

        System.out.println(str1);
        System.out.println(str2);
        System.out.println(str3);
        System.out.println(str4);
        System.out.println(str5);
        System.out.println(str6);
   }
}
```

```
101
|            101|
|Hello, Java     |
|00000101.000000|
|          101.00|
65
```

ARRAY

Arrays

- int [] intArray = new int[10] ;
  - An array of size 10
  - Index starts at 0.




```
class ArrayExample {
    public static void main(String[] args) {
        int [] ia = {0, 1, 2, 3} ;
        for (int i = 0; i < ia.length; i++)
            System.out.println(ia[i]);
    }
}
```

Arrays: An Example

```
import java.util.Random;

public class AnotherArrayExample {
   public static void main(String[] args) {
      Random oRandom = new Random() ;

      int [] ia = new int[101];
      for (int i = 0; i < ia.length; i++) {
         ia[i] = oRandom.nextInt(100) ;
         System.out.println(ia[i]) ;
      }

      int sum = 0;
      for (int v: ia) // for each loop
         sum += v;
      System.out.println(sum);
   }
}
```

Copying Arrays

- Shallow copy

```
int [] smallPrimes = {2, 3, 5, 7, 11, 13} ;
int [] luckyNumbers = smallPrimes ;
luckyNumbers[5] = 12 ; // now smallPrimes[5] is also 12
```

- Deep copy: System.arraycopy(from, fromIndex, to, toIndex, count) ;

```
class ArrayCopy {
  public static void main(String args[]) {
    int [] smallPrimes = {2, 3, 5, 7, 11, 13} ;
    int [] luckyNumbers = {1001, 1002, 1003, 1004, 1005, 1006, 1007};
    System.arraycopy(smallPrimes, 2, luckyNumbers, 3, 4) ;
    for ( int v : luckyNumbers )
      System.out.print(v + " ") ;                    // 1001 1002 1003 5 7 11 13
  }
}
```

Arrays Class

- java.util.Arrays class provides useful array operations.

```
public class ArraysExample {
    public static void main(String[] args) {
        int[] array1 = new int[10];
        for(int i = 0; i < array1.length; i++) array1[i] = i;
        System.out.println(Arrays.binarySearch(array1, 7)); // 7

        int[] array2 = Arrays.copyOf(array1, 10);
        for (int v: array2) System.out.print(v + " "); // 0 1 2 3 4 5 6 7 8 9
        System.out.println(Arrays.equals(array1, array2)); // true

        int[] array3 = Arrays.copyOfRange(array1, 2, 5);
        System.out.println();
        for (int v: array3) System.out.print(v + " "); // 2 3 4
        System.out.println(Arrays.equals(array1, array3)); // false

        int[] array4 = new int[5];
        Arrays.fill(array4, 7);
        System.out.println();
        for (int v: array4) System.out.print(v + " "); // 7 7 7 7 7
    }
}
```

int [] array vs int array []

- Q: int [] ia and int ia [] are same ?
- A: They are different! Use int [] ia rather than int ia[].

```
class ArrayInit {
    public static void main(String args[]) {
        int[] a1 = {10, 20, 30}, a2 = {100, 200, 300} ;
        int a3[] = new int[10], a4 = a1 ;
        // ERROR: incompatible types, found: int[], required: int
    }
}
```

ENUM

Enumerated Type: enum

- Enumerated type is used to specify a variable with a limited set of values.

```
enum Fruit {APPLE, GRAPE, PEAR} ;
public class EnumExample1 {
   public static void main(String[] args) {
      Fruit myFruit = Fruit.APPLE ; // Fruit.valueOf("APPLE")
      System.out.println(myFruit) ;

      String apple = getFruitName(myFruit);
      System.out.println("The fruit is " + apple ) ;

      String grape = getFruitName(Fruit.valueOf("GRAPE"));
      System.out.println("The fruit is " + grape ) ;
   }
```

```
APPLE
The fruit is 사과
The fruit is 포도
```

Enumerated Type: enum

```
    private static String getFruitName(final Fruit myFruit) {
        String fruitName ;
        switch ( myFruit ) {
            case APPLE : fruitName = "사과" ; break ;
            case GRAPE : fruitName = "포도" ; break ;
            case PEAR : fruitName = "배" ; break ;
            default : fruitName = "모름" ; break ;
        }
        return fruitName;
    }
}
```

Enumerated Type: enum

- You can specify values of enum constants at the creation time

```
enum Currency {
   PENNY(1), NICKLE(5), DIME(10), QUARTER(25);
   private int value;
   private Currency(int value) {
      this.value = value;
   }
}
public class EnumExample2 {
   public static void main(String args[]) {
      Currency usCoin = Currency.DIME;
      if ( usCoin == Currency.DIME ) {
         System.out.println("enum in java can be compared using ==");
      }
      for ( Currency coin: Currency.values() ) {
         System.out.println("coin: " + coin);
      }
   }
}
```

```
enum in java can be compared using ==
coin: PENNY
coin: NICKLE
coin: DIME
coin: QUARTER
```

String in Switch Case

- Since JDK 7(2011), String is allowed in the expression of a switch statement

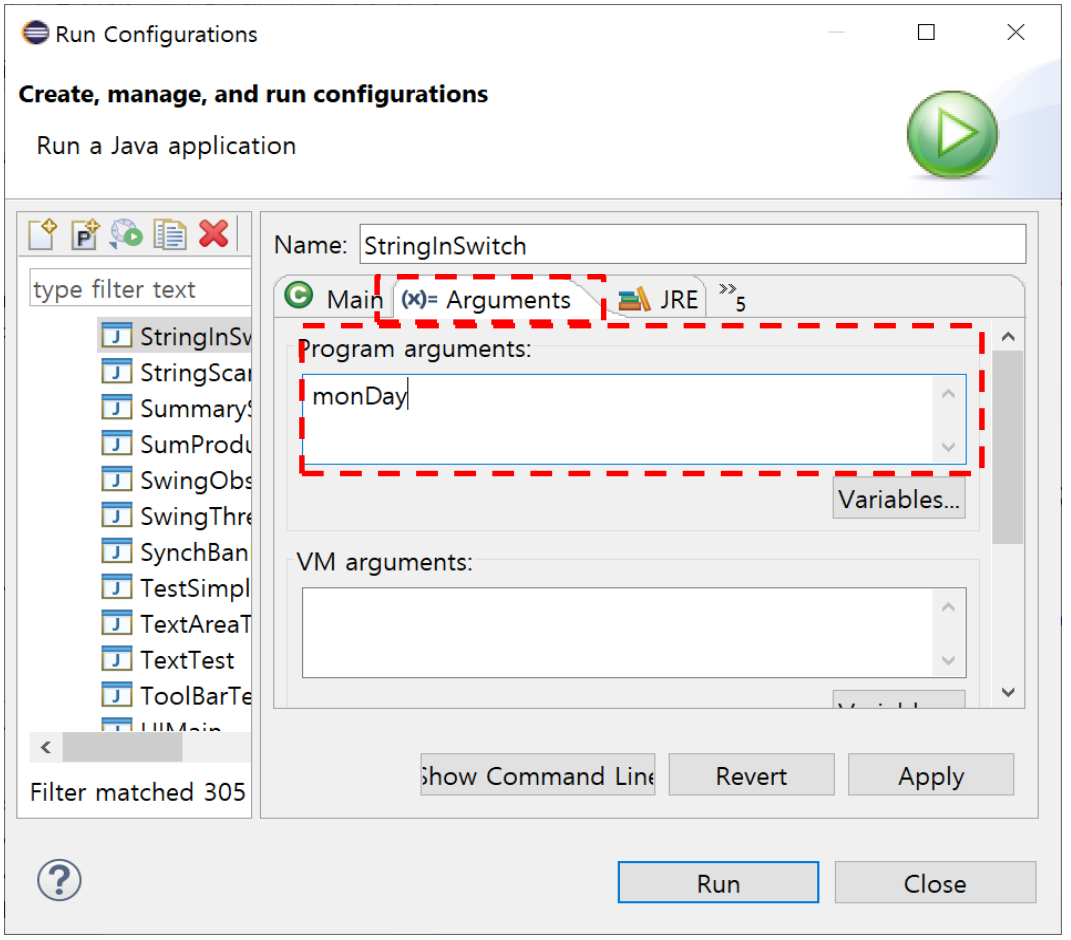
```
public class StringInSwitch {
 public static void main(String[] args) {
   String dayOfWeekArg = args[0];
   String typeOfDay = getTypeOfDay(dayOfWeekArg);
   System.out.printf("%10s is %20s%n", dayOfWeekArg, typeOfDay);
 }
```

String in Switch Case

```
private static String getTypeOfDay(String dayOfWeekArg) {
  String typeOfDay;
  switch ( dayOfWeekArg.toUpperCase() ) {
    case "MONDAY": typeOfDay = "Start of work week"; break;
    case "TUESDAY":
    case "WEDNESDAY":
    case "THURSDAY": typeOfDay = "Midweek"; break;
    case "FRIDAY": typeOfDay = "End of work week"; break;
    case "SATURDAY":
    case "SUNDAY": typeOfDay = "Weekend"; break;
    default:
      throw new IllegalArgumentException("Invalid day of the week: " + dayOfWeekArg);
  }
  return typeOfDay;
 }
}
```

Program Arguments in Eclipse

- Run As – Run Configurations…



Constants

- You can use the keyword **final** to denote a constantness for local variable and parameter

```
import java.util.Scanner;
public class FinalVariableParameter {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in) ;
      final int n = scanner.nextInt() ;
      scanner.close() ;
      // n = 200 ; final local variable cannot be assigned!
      System.out.printf("Factorial of " + n + ": %,20d", factorial(n)) ;
   }
   public static long factorial(final int v) {
      // v = 100 ; final local variable cannot be assigned!
      long result = 1 ;
      for ( int i = 2 ; i <= v ; i ++ ) result *= i ;
      return result ;
   }
}
```

INPUT AND OUTPUT

Reading Input by Scanner

- Scanner class is used to read typed values from the console

```
public class ScannerExample1 {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      // get first input
      System.out.print("What is your name? ");
      String name = scanner.nextLine();
      // get second input
      System.out.print("How old are you? ");
      int age = scanner.nextInt();
      // display output on console
      System.out.println("Hello, " + name + ". Next year, you'll be " + (age+1));

      scanner.close();
   }
}
```

```
What is your name? Kim
How old are you? 20
Hello, Kim. Next year, you'll be 21
```

Scanner

- Major methods in Scanner Class

method	description
String nextLine()	Reads the next **line** of input
String next()	Reads the next **word** of input (delemited by whitespace)
int nextInt()	Read the next **integer**.
double nextFloat() double nextDouble()	Read the next **floating point number**
boolean hasNext()	**Tests** whether there is another **word** in the input
boolean hasNextInt()	**Tests** whether the next word represents an **integer**
boolean hasNextDouble()	**Tests** whether the next word represents a **floating-point number**

Reading Input by Scanner

```
import java.util.Scanner;
public class ScannerExample2 {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in) ;

      System.out.println("Enter two integers!") ;
      final int n1 = scanner.nextInt();
      final int n2 = scanner.nextInt() ;
      System.out.println("Enter operator: [+, -] !") ;
      final String strOp = scanner.next() ;
      scanner.close() ;

      final char charOp = strOp.charAt(0) ;
      int result ;
      switch ( charOp ) {
         case '+' : result = n1 + n2 ; break ;
         case '-' : result = n1 - n2 ; break ;
         default: result = 0 ; break ;
      }
      System.out.println(result) ;
   }
}
```

```
Enter two integers!
200 400
Enter operator: [+, -] !
+
600
```

Scanner from String

- Scanner can be constructed from String

```
public class StringScanner {
    public static void main(String[] args) {
        final String message = "Hello World₩nWelcom Java!";
        Scanner scanner = new Scanner(message);

        while ( scanner.hasNext() ) {
            final String word = scanner.next();
            System.out.println(word);
        }
        scanner.close();
    }
}
```

```
Hello
World
Welcom
Java!
```

InputMismatchException

```
1:   import java.util.Scanner;
2:   public class ScannerExample3 {
3:       public static void main(String[] args) {
4:
5:       Scanner scanner = new Scanner(System.in) ;
6:       while ( scanner.hasNext()) {
7:           final int n = scannerObject.nextInt() ;
8:           System.out.println(n) ;
9:       }
10:      scanner.close() ;
11:  }
12:}
100
100
100F
Exception in thread "main" java.util.InputMismatchException
            at java.util.Scanner.throwFor(Unknown Source)
            at java.util.Scanner.next(Unknown Source)
            at java.util.Scanner.nextInt(Unknown Source)
            at java.util.Scanner.nextInt(Unknown Source)
            at ScannerExample2.main(ScannerExample3.java:7)
```

"100F" cannot be translated into an Integer

Catching InputMismatchException

- How can we handle exceptions in our own way?
- Let's catch the exceptions in our code!

```
import java.util.Scanner;
public class ScannerException {
   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in) ;
      try {
         while ( scanner.hasNext()) {
            final int n = scanner.nextInt() ;
            System.out.println(n) ;
         }
      } catch (Exception e) {
         System.out.println("Exception: " + e) ;
         System.out.println("정수 형태의 문자열을 입력하세요!") ;
      }
      finally { scanner.close() ; }
   }
}
```

```
100
100
100F
Exception: java.util.InputMismatchException
정수 형태의 문자열을 입력하세요!
```

Formatting Output

- Like printf() in C++, you can use printf in Java.

Converter	Description	Example
%s	String	Hello
%c	Character	H
%d	Decimal integer	159
%o	Octal integer	237
%x	Hexadecimal integer	9f
%f	Fixed-point Floating point number	15.9
%e	Exponential floating point	1.59e+01
%b	boolean	true
%n	New line. Use this instead of ₩n	

Formatting Output

- Flags used to control the appearance of the formatted output.
  - System.out.printf("%**,.2**f", 10000.0 / 3.0) prints 3,333.33

Flag	Description	Example
+	Print sign character	+3333.33
0	Add leading zeros	003333.33
-	Left-justify field	3333.33
(	Enclose negative number in parentheses	(3333.33)
,	Add group separator	3,3333.33
# (for x or o)	Add 0x or 0 prefix	0xcafe
$	Specify the index of the argument to be formatted. %**1$**d %**2$**x	

Formatting Output: Example

```
public class FormatTest {

   public static void main(String[] args) {
    long n = 123456;

    System.out.printf("%d%n", n);
    System.out.printf("%10d%n", n);            // width
    System.out.printf("%-10d%n", n);           // left-justified
    System.out.printf("%010d%n", n);           // leading zeroes
    System.out.printf("%+10d%n", n);           // sign character
    System.out.printf("%,10d%n", n);           // group character
    System.out.format("%d₩t%1$#x%n%n", n); // argument index and hexadecimal

    double pi = Math.PI;
    System.out.printf("%n%f%n", pi);            // fixed-point format
    System.out.printf("%e%n", pi);              // exponential format
    System.out.printf("%10.3f%n", pi);     // width/precision in fixed-point format
    System.out.printf("%10.3e%n", pi);    // width/precision in exponential format
    System.out.printf("%+-10.3f%n", pi); // sign character and left-justified
   }
}
```

```
123456
    123456
123456
0000123456
   +123456
    123,456
123456   0x1e240

3.141593
3.141593e+00
     3.142
 3.142e+00
+3.142
```

DATE & TIME

Getting Current Date and Time

```
import java.util.Date;

public class CurrentDateTime {
   public static void main(String[] args) {
      Date date = new Date();
      System.out.println(date.toString());
   }
}
```

```
Sun Sep 08 00:36:14 KST 2019
```

Date Formatting Using SimpleDateFormat

```
import java.text.SimpleDateFormat;
import java.util.Date;

public class DateFormat {

   public static void main(String[] args) {
      Date now = new Date( );
      SimpleDateFormat format =
            new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");

      System.out.println("Current Date: " + format.format(now));
   }
}
```

Current Date: 일 2019.09.08 at 12:37:50 오전 KST

Sleeping for a While

```
import java.util.Date;

public class Sleep {
   public static void main(String[] args) {
      try {
         System.out.println(new Date( ));
         Thread.sleep(3 * 1000); // 3 seconds
         System.out.println(new Date( ));
      } catch (Exception e) {
         System.out.println("Got an exception!");
      }
   }
}
```

```
Sun Sep 08 00:39:06 KST 2019
Sun Sep 08 00:39:09 KST 2019
```

Measuring Elapsed Time

```
import java.util.Date;

public class ElapsedTimeMeasure {
   public static void main(String[] args) {
      try {
            final long start = System.currentTimeMillis();
            System.out.println(new Date( ));

            Thread.sleep(3 * 1000);
            System.out.println(new Date( ));

            final long end = System.currentTimeMillis();
            System.out.println("Difference is : " + (end - start));
         } catch (Exception e) {
            System.out.println("Got an exception!");
         }
   }
}
```

```
Sun Sep 08 00:42:44 KST 2019
Sun Sep 08 00:42:47 KST 2019
Difference is : 3037
```