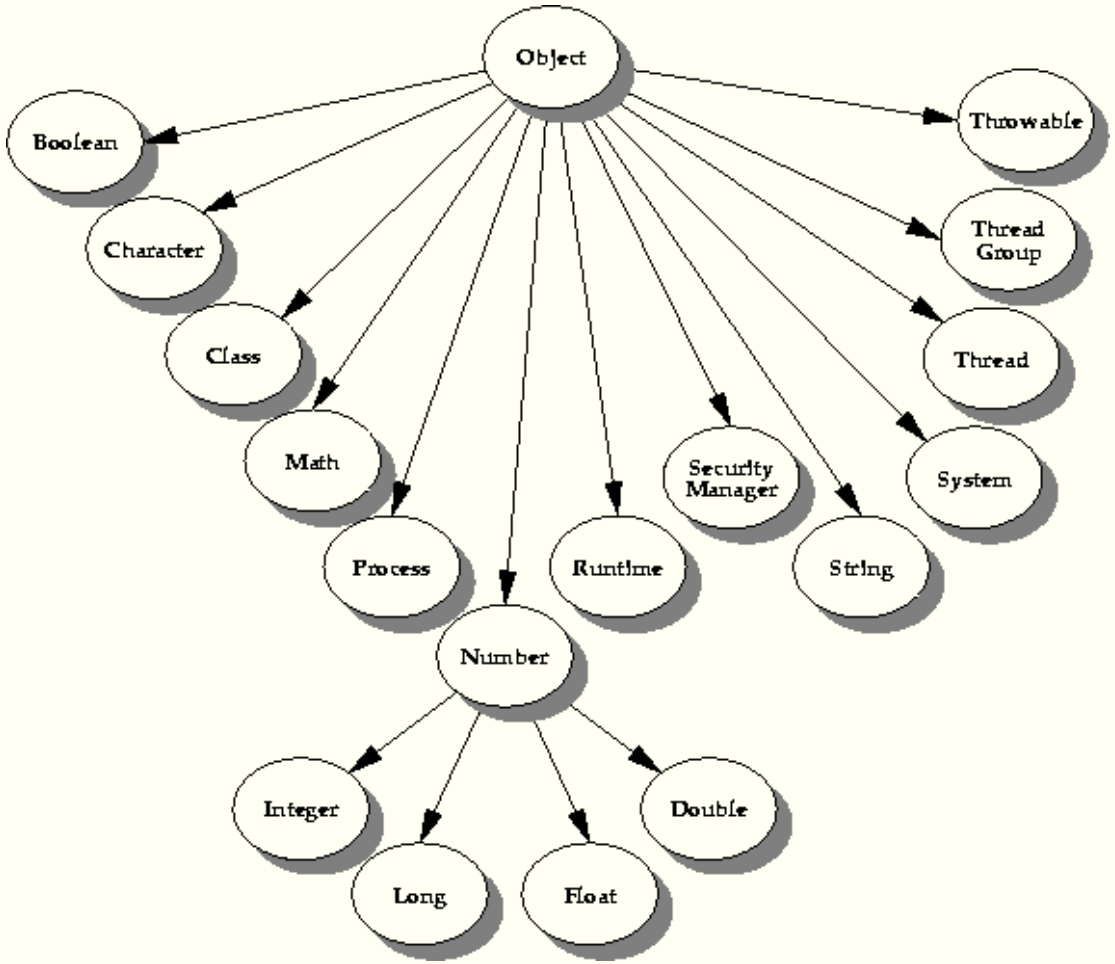
ПОСИЛАЛЬНІ ТИПИ ДАНИХ ДЛЯ МАТЕМАТИЧНИХ ОБЧИСЛЕНЬ

Питання 2.2.

Базова ієрархія класів Java API



- Базові API описані в бібліотеках, наприклад, java.lang.
  - Клас Math описує константи E = 2.718281828459045 та PI = 3.141592653589793 типу double, що відповідають значенням експоненти та числа π.

Клас java.lang.Math

Метод	Опис

double abs(double d);
float abs(float f);
int abs(int i);
long abs(long l)
	Повертає абсолютне значення (модуль) d. Особливі випадки: abs(-0.0) = +0.0, abs(-infinity) = +infinity, abs(NaN) = NaN, abs(Integer.MIN_VALUE) = Integer.MIN_VALUE, abs(Long.MIN_VALUE) = Long.MIN_VALUE,

double acos(double d)	Повертає значення arccos(d) у діапазоні від 0 до PI. Особливі випадки: acos(d > 1) = NaN, acos(d < -1) = NaN, acos(NaN) = NaN
double asin(double d)	Повертає значення arcsin(d) у діапазоні від –PI/2 до PI/2. Особливі випадки: asin(d > 1) = NaN, asin(d < -1) = NaN, asin(NaN) = NaN
double atan(double d)	Повертає значення arctg(d) у діапазоні від –PI/2 до PI/2. Особливі випадки: atan(+0.0) = +0.0, atan(-0.0) = -0.0, atan(+infinity) = +PI/2, atan(-infinity) = -PI/2, atan(NaN) = NaN,
double ceil(double d)	Повертає найменше значення (найближче до -∞), яке не менше за d та ціле. Особливі випадки: ceil(+0.0) = +0.0, ceil(-0.0) = -0.0, ceil(-1.0<d<0.0) = -0.0, ceil(+infinity) = +infinity, ceil(-infinity) = -infinity, ceil(NaN) = NaN

Клас java.lang.Math

Метод	Опис
double cos(double d)	Повертає cos(d), d – в радіанах. Особливі випадки: cos(+infinity) = NaN, cos(-infinity) = NaN, cos(NaN) = NaN.
double exp(double d)	Повертає значення e^d. Особливі випадки: exp(+infinity) = +infinity, exp(-infinity) = +0.0, exp(NaN) = NaN
double floor(double d)	Повертає найбільше значення (найближче до +∞), яке не більше за d та ціле. Особливі випадки: floor(+0.0) = +0.0, floor(-0.0) = -0.0, floor(+infinity) = +infinity, floor(-infinity) = -infinity, floor(NaN) = NaN
double log(double d)	Повертає ln(d). Особливі випадки: log(+0.0) = -infinity, log(-0.0) = -infinity, log(d < 0) = NaN, log(+infinity) = +infinity, log(-infinity) = NaN, log(NaN) = NaN
double log10(double d)	Повертає $\log_{10}(d)$. Особливі випадки: log10(+0.0) = -infinity, log10(-0.0) = -infinity, log10(d < 0) = NaN, log10(+infinity) = +infinity, log10(-infinity) = NaN, log10(NaN) = NaN

Клас java.lang.Math

Метод	Опис

double max(double d1, double d2);
float max(float f1, float f2);
int max(int i1, int i2);
long max(long l1, long l2)
	Повертає найближче до +∞ значення серед переданих.
	Особливі випадки для дробових чисел:
	max(NaN, число) = NaN, max(число, NaN) = NaN, max(+0.0, -0.0) = +0.0, max(-0.0, +0.0) = +0.0.

double min(double d1, double d2);
float min(float f1, float f2);
int min(int i1, int i2);
long min(long l1, long l2)
	Повертає найближче до −∞ значення серед переданих.
	Особливі випадки для дробових чисел:
	min(NaN, число) = NaN, min(число, NaN) = NaN, min(+0.0, -0.0) = -0.0, min(-0.0, +0.0) = -0.0.

double random()	Повертає псевдовипадкове число в діапазоні [0.0, 1.0)
int round(long roun	long round(double d) — Returns the result of rounding d to a long integer. The result is equivalent to (long) Math.floor(d + 0.5). There are seven special cases: round(+0.0) = +0.0, round(-0.0) = +0.0, round(anything > Long.MAX_VALUE) = Long.MAX_VALUE, round(anything < Long.MIN_VALUE) = Long.MIN_VALUE, round(+infinity) = Long.MAX_VALUE, round(-infinity) = Long.MIN_VALUE, and round(NaN) = +0.0. int round(float f) — Returns the result of rounding f to an integer. The result is equivalent to (int) Math.floor(f + 0.5). There are seven special cases: round(+0.0) = +0.0, round(-0.0) = +0.0, round(anything > Integer.MAX_VALUE) = Integer.MAX_VALUE, round(anything < Integer.MIN_VALUE) = Integer.MIN_VALUE, round(+infinity) = Integer.MAX_VALUE, round(-infinity) = Integer.MIN_VALUE, and round(NaN) = +0.0.	...Math.floor(d+0.5) та ...и: round(+0.0) = +0.0, ...) = Long.MAX_VALUE,

Клас java.lang.Math

`double signum(double d)`	Returns the sign of `d` as -1.0 (`d` less than 0.0), 0.0 (`d` equals 0.0), and 1.0 (`d` greater than 0.0). There are five special cases: `signum(+0.0) = +0.0`, `signum(-0.0) = -0.0`, `signum(+infinity) = +1.0`, `signum(-infinity) = -1.0`, and `signum(NaN) = NaN`.
`float signum(float f)`	Returns the sign of `f` as -1.0 (`f` less than 0.0), 0.0 (`f` equals 0.0), and 1.0 (`f` greater than 0.0). There are five special cases: `signum(+0.0) = +0.0`, `signum(-0.0) = -0.0`, `signum(+infinity) = +1.0`, `signum(-infinity) = -1.0`, and `signum(NaN) = NaN`.
`double sin(double d)`	Returns the sine of angle `d` (expressed in radians). There are five special cases: `sin(+0.0) = +0.0`, `sin(-0.0) = -0.0`, `sin(+infinity) = NaN`, `sin(-infinity) = NaN`, and `sin(NaN) = NaN`.
`double sqrt(double d)`	Returns the square root of `d`. There are five special cases: `sqrt(+0.0) = +0.0`, `sqrt(-0.0) = -0.0`, `sqrt(anything < 0) = NaN`, `sqrt(+infinity) = +infinity`, and `sqrt(NaN) = NaN`.

Клас java.lang.Math

`double tan(double d)`	Returns the tangent of angle `d` (expressed in radians). There are five special cases: `tan(+0.0) = +0.0`, `tan(-0.0) = -0.0`, `tan(+infinity) = NaN`, `tan(-infinity) = NaN`, and `tan(NaN) = NaN`.
`double toDegrees(double angrad)`	Converts angle `angrad` from radians to degrees via expression `angrad * 180 / PI`. There are five special cases: `toDegrees(+0.0) = +0.0`, `toDegrees(-0.0) = -0.0`, `toDegrees(+infinity) = +infinity`, `toDegrees(-infinity) = -infinity`, and `toDegrees(NaN) = NaN`.
`double toRadians(double angdeg)`	Converts angle `angdeg` from degrees to radians via expression `angdeg / 180 * PI`. There are five special cases: `toRadians(+0.0) = +0.0`, `toRadians(-0.0) = -0.0`, `toRadians(+infinity) = +infinity`, `toRadians(-infinity) = -infinity`, and `toRadians(NaN) = NaN`.

- Math.abs() повертає абсолютне значення переданого числа (цілого чи дробового).
  - Проте також його часто використовують для порівняння чисел.
  - У зв'язку з неточним представленням дробових чисел в системі, зокрема, вираз 0.3==0.1+0.1+0.1 набуде значення false.
  - За допомогою методу abs() та певного значення похибки вираз Math.abs(0.3 - (0.1 + 0.1 + 0.1)) < 0.1 буде мати значення true, оскільки абсолютна відстань набагато менше заданої похибки.

Метод random()

- Метод random() генерує псевдовипадкове число.
  - Діапазон чисел від 0.0 до 1.0 часто непрактичний, і відповідні результати масштабують на потрібний діапазон.
  - Наприклад, реалізація методу

```
static int rnd(int limit)
{
   return (int) (Math.random() * limit);
}
```

  - дає можливість генерувати випадкові числа від 0 до limit-1.
  - Інструкція -100 + rnd(201) дозволяє отримувати випадкові числа з діапазону [-100, 100].
- Запис (int) Math.random() * limit завжди буде давати 0, оскільки спочатку відбувається зведення типу числа згенерованого random() шляхом усікання дробової частини.

Клас StrictMath та зарезервоване слово strictfp

- Даний клас знаходиться в пакеті java.lang та майже ідентичний класу Math.
- Методи StrictMath повертають однакові результати на всіх платформах, у той час, як деякі методи класу Math можуть повертати значення, що будуть відрізнятись від платформи до платформи.
  - StrictMath не підтримує специфічні для платформ риси, наприклад, математичний співпроцесор.
- У більшості випадків методи класу Math викликають свої відповідники з класу StrictMath.
  - Виняток: методи toDegrees() та toRadians().
  - Хоч тіло цих методів ідентичне, реалізація з класу StrictMath включає зарезервоване слово strictfp в заголовку:
  - public static strictfp double toDegrees(double angrad)
  - public static strictfp double toRadians(double angdeg)

Strictfp - для обрізання точності обчислень з метою підтримки портативності

- Дає підтримку портативності в контексті проміжних floating-point представлень чисел з плаваючою крапкою та їх переповнень (overflow) чи недозаповнень (underflow).
  - Без strictfp проміжні обчислення не обмежуються IEEE 754: точність обчислень залежить від платформи.
- Застосоване до методу ключове слово strictfp задає стиль виконання всіх обчислень з плаваючою крапкою, що виконуються в методі.
  - Проте strictfp можна використовувати і в заголовку оголошення класу, щоб забезпечити строге виконання обчислень з плаваючою крапкою в цьому класі.
  - Наприклад, `public strictfp class FourierTransform`

Клас Number

- Абстрактний клас java.lang.Number є суперкласом для тих класів, які представляють числові значення та можуть конвертуватись у примітивні типи.

- Методи для конвертації
  - byte byteValue()
  - double doubleValue()
  - float floatValue()
  - int intValue()
  - long longValue()
  - short shortValue()

- java.util.concurrent.atomic.AtomicInteger
- java.util.concurrent.atomic.AtomicLong
- java.math.BigDecimal
- java.math.BigInteger
- java.lang.Byte
- java.lang.Double
- java.lang.Float
- java.lang.Integer
- java.lang.Long
- java.lang.Short

Клас BigDecimal

```
public class SavingsAccount
{
   int balance;

   SavingsAccount deposit(int amount)
   {
      balance += amount;
      return this;
   }

   SavingsAccount printBalance()
   {
      System.out.println(balance);
      return this;
   }
  public static void main(String[] args)
  {
     new SavingsAccount().deposit(1000).printBalance();
  }
}
```

- Клас описує виконання внеску та друк балансу накопичувального рахунку.
- Баланс – ціле число!
  - Причина в дробовій частині, яка часто не має точного представлення (замість 5.1 отримується 5.099999999999998)
  - Кожне обчислення з плаваючою комою потрібно округлювати, інакше будуть виникати невеликі похибки.
    - Похибки незначні, проте округлення до найближчої копійки може призвести до некоректного результату
    - + Проблеми форматування виводу
  - Баланс вноситься в копійках, проте для великих сум потрібні великі числа!

Похибки сервісів General Ledger

- У багатьох місцях проявлялись помилки порядку сотої відсотка або більше.
  - Облік грошей стає майже неможливим.
  - Хто хотів би виставити рахунок клієнту на $4.01, коли в замовленні $4.00?
- Причини: обчислення, які містять суми, кількості, корективи тощо зазвичай виконуються з незначною увагою до проблем точності та округлення, які виникають при роботі з фінансами.
  - Всі ці обчислення використовували тип double, який не передбачає способу керування округленням числа або обмеження точності в обчисленнях.
  - Альтернатива – клас java.math.BigDecimal.
- Припустимо, є продукт вартістю 10 у. о., а місцевий податок складає 8.25%.
  - На папері сума відрахувань складає: 10.00 * 0.0825 = 0.825
  - Оскільки точність обчислень для валюти – 2 знаки після коми, треба округлювати 0.825 до копійки в більшу сторону. Податок стане 0.825 -> 0.83
- Клієнту прийде рахунок на 10.83 у.о., а податківець отримає 0.83 у.о.
  - При 1000 проданих продуктах переплата податку склала б: 1000 * (0.83 - 0.825) = 5.0

Де виконувати округлення в обчисленні?

- Припустимо, рідкий азот коштує 0.528361 за літр.
  - Клієнт купує 100.00 літрів: 100.0 * 0.528361 = 52.8361
  - Оскільки це не податок, можна округлити стандартно: 52.84.
  - Нехай ми хочемо дати дисконтну скидку 5% від покупки.
  - З якої суми робити знижку?
    1) 52.8361 * 0.95 = 50.194295 = 50.19;
    2) 52.84 * 0.95 = 50.198 = 50.20
- Старий код завжди виконував обчислення типу 1.
  - Але в новому коді перед обчисленям знижок, податків та ін. спочатку виконується округлення, як у випадку 2.
  - Це одна з головних причин для помилки в один цент.
- Необхідні дві речі (є у класі java.math.BigDecimal):
  - Можливість задати масштаб (кількість цифр після десяткової крапки)
  - Можливість задати метод округлення.

Конструктори та методи класу BigDecimal

Метод	Опис
BigDecimal(int val)	За умовчанням val = 0
BigDecimal(int val)	Ініціалізує еквівалент десяткового значення. Викидає виняток java.lang.NullPointerException, коли val = null та java.lang.NumberFromatException, коли рядкове представлення числа некоректне.
BigDecimal abs()	Повертає новий об'єкт BigDecimal, який містить абсолютне значення поточного об'єкта.
BigDecimal add(BigDecimal augend)	Повертає новий об'єкт BigDecimal, який містить суму значень поточного об'єкта та об'єкта augend. Викидає NPE, коли augend = null
BigDecimal divide(BigDecimal divisor)	Повертає новий об'єкт BigDecimal, який містить частку від ділення значення поточного об'єкта на значення об'єкта divisor. Викидає NPE, коли divisor = null та java.lang.ArithmeticException, коли divisor = 0 або результат неможливо точно представити.

`BigDecimal max(BigDecimal val)`	Returns either `this` or `val`, whichever `BigDecimal` instance contains the larger value. This method throws `NullPointerException` when `val` is `null`.
`BigDecimal min(BigDecimal val)`	Returns either `this` or `val`, whichever `BigDecimal` instance contains the smaller value. This method throws `NullPointerException` when `val` is `null`.
`BigDecimal multiply(BigDecimal multiplicand)`	Returns a new `BigDecimal` instance that contains the product of the current value and the argument value. The resulting scale is the sum of the current and argument scales. This method throws `NullPointerException` when `multiplicand` is `null`.
`BigDecimal negate()`	Returns a new `BigDecimal` instance that contains the negative of the current value. The resulting scale is the same as the current scale.
`int precision()`	Returns the precision of the current `BigDecimal` instance.
`BigDecimal remainder(BigDecimal divisor)`	Returns a new `BigDecimal` instance that contains the remainder of the current value divided by the argument value. The resulting scale is the difference of the current scale and the argument scale. It might be adjusted when the result requires more digits. This method throws `NullPointerException` when `divisor` is `null` or `ArithmeticException` when `divisor` represents 0.
`int scale()`	Returns the scale of the current `BigDecimal` instance.

Method	Description
`BigDecimal setScale(int newScale, RoundingMode roundingMode)`	Returns a new `BigDecimal` instance with the specified scale and rounding mode. If the new scale is greater than the old scale, additional zeros are added to the unscaled value. In this case, no rounding is necessary. If the new scale is smaller than the old scale, trailing digits are removed. If these trailing digits are not zero, the remaining unscaled value has to be rounded. For this rounding operation, the specified rounding mode is used. This method throws `NullPointerException` when `roundingMode` is `null`, and `ArithmeticException` when `roundingMode` is set to `RoundingMode.ROUND_UNNECESSARY` but rounding is necessary based on the current scale.
`BigDecimal subtract(BigDecimal subtrahend)`	Returns a new `BigDecimal` instance that contains the current value minus the argument value. The resulting scale is the maximum of the current and argument scales. This method throws `NullPointerException` when `subtrahend` is `null`.
`String toString()`	Returns a string representation of this `BigDecimal` instance. Scientific notation is used when necessary.

Константи режимів округлення

- Метод setScale(scale) задає кількість цифр після коми.
  - Проте хорошою практикою є одночасне вказування разом з масштабом режиму округлення за допомогою setScale(scale, roundingMode).
  - Режим округлення задає правило округлення числа.

Константа	Опис
CEILING	Округлює в сторону $+\infty$
DOWN	Округлює вбік нуля
FLOOR	Округлює в сторону $-\infty$
HALF_DOWN	Округлює до «найближчого сусіда», при рівній відстані – до меншого.
HALF_EVEN	Округлює до «найближчого сусіда», при рівній відстані – до парного.
HALF_UP	Округлює до «найближчого сусіда», при рівній відстані – до більшого.
UNNECESSARY	Усуває округлення, якщо результат точний
UP	Додатні значення округлюються вбік $+\infty$, а від'ємні – в сторону $-\infty$

Незмінюваність та арифметика

- Числа **BigDecimal** є незмінюваними.
  - Якщо створюється новий об’єкт **BigDecimal** із значенням "2.00", воно залишиться "2.00" та ніколи не зміниться.
- Як же виконуються математичні обчислення?
  - Методи **.add()**, **.multiply()** та інші повертають новий об’єкт **BigDecimal**, який містить результат.
  - Наприклад, щоб отримати результат при обчисленні суми:

    amount = amount.add(thisAmount);
  - Переконайтесь, що це робить не так:

    amount.add(thisAmount);
  - Це найбільш поширена помилка при роботі з BIGDECIMAL!

Порівняння

- Ніколи не використовуйте для порівняння **BigDecimal** метод **.equals()**.
  - **equals** буде порівнювати масштаби: поверне хибу, навіть якщо математично числа будуть рівними:

```
BigDecimal a = new BigDecimal("2.00");
BigDecimal b = new BigDecimal("2.0");
print(a.equals(b));  // хиба
```

- Потрібно використовувати методи **.compareTo()** та **.signum()**.

```
a.compareTo(b);      // повертає (-1 якщо a < b),
      (0 якщо a == b), (1 якщо a > b)
a.signum();  // повертає (-1 якщо a < 0),
      (0 якщо a == 0), (1 якщо a > 0)
```

Клас BigInteger та його конструктори

- BigInteger - незмінюваний клас, який представляє знакове ціле довільної точності, що зберігається в екземплярі класу.
- BigInteger оголошує 3 константи: ONE, TEN, ZERO.
  - BigInteger-еквіваленти 1, 10 та 0.

Method	Description
`BigInteger(byte[] val)`	Initializes the `BigInteger` instance to the integer that is stored in the `val` array, with `val[0]` storing the integer's most significant (leftmost) 8 bits. This constructor throws `NullPointerException` when `val` is `null` and `NumberFormatException` when `val.length` equals 0.
`BigInteger(String val)`	Initializes the `BigInteger` instance to the integer equivalent of `val`. This constructor throws `NullPointerException` when `val` is `null` and `NumberFormatException` when `val`'s string representation is invalid (contains letters, for example).

`BigInteger abs()`	Returns a new `BigInteger` instance that contains the absolute value of the current instance's value.
`BigInteger add(BigInteger augend)`	Returns a new `BigInteger` instance that contains the sum of the current value and the argument value. This method throws `NullPointerException` when `augend` is `null`.
`BigInteger divide(BigInteger divisor)`	Returns a new `BigInteger` instance that contains the quotient of the current value divided by the argument value. This method throws `NullPointerException` when `divisor` is `null` and `ArithmeticException` when `divisor` represents 0 or the result cannot be represented exactly.
`BigInteger max(BigInteger val)`	Returns either `this` or `val`, whichever `BigInteger` instance contains the larger value. This method throws `NullPointerException` when `val` is `null`.
`BigInteger min(BigInteger val)`	Returns either `this` or `val`, whichever `BigInteger` instance contains the smaller value. This method throws `NullPointerException` when `val` is `null`.
`BigInteger multiply(BigInteger multiplicand)`	Returns a new `BigInteger` instance that contains the product of the current value and the argument value. This method throws `NullPointerException` when `multiplicand` is `null`.
`BigInteger negate()`	Returns a new `BigInteger` instance that contains the negative of the current value.
`BigInteger remainder(BigInteger divisor)`	Returns a new `BigInteger` instance that contains the remainder of the current value divided by the argument value. This method throws `NullPointerException` when `divisor` is `null` and `ArithmeticException` when `divisor` represents 0.
`BigInteger subtract(BigInteger subtrahend)`	Returns a new `BigInteger` instance that contains the current value minus the argument value. This method throws `NullPointerException` when `subtrahend` is `null`.
`String toString()`	Returns a string representation of this `BigInteger` instance.

Приклад використання

```
import java.math.BigInteger;

public class FactComp
{
   public static void main(String[] args)
   {
      System.out.println(factorial(12));
      System.out.println();
      System.out.println(factorial(20L));
      System.out.println();
      System.out.println(factorial(170.0));
      System.out.println();
      System.out.println(factorial(new BigInteger("170")));
      System.out.println();
      System.out.println(factorial(25.0));
      System.out.println();
      System.out.println(factorial(new BigInteger("25")));
   }
```

```
   static int factorial(int n)
   {
      if (n == 0)
         return 1;
      else
         return n * factorial(n - 1);
   }

   static long factorial(long n)
   {
      if (n == 0)
         return 1;
      else
         return n * factorial(n - 1);
   }

   static double factorial(double n)
   {
      if (n == 1.0)
         return 1.0;
      else
         return n * factorial(n - 1);
   }

   static BigInteger factorial(BigInteger n)
   {
      if (n.equals(BigInteger.ZERO))
         return BigInteger.ONE;
      else
         return n.multiply(factorial(n.subtract(BigInteger.ONE)));
   }
}
```

Вивід застосунку:

```
479001600

2432902008176640000

7.257415615307994E306

725741561530799896739672821112926311471699168129645137654357779890056184340170615785235074924261745
951149099123783852077666602256544275302532890077320751090240043028005829560396661259965825710439855
829425756896631343961226257109494680671120556888045719334021266145280000000000000000000000000000000
0000000000

1.5511210043330986E25

155112100433309859840000000
```

Класи-обгортки примітивних типів

- Класи Byte, Double, Float, Integer, Long, Short, Boolean та Character відомі як ***класи-обгортки*** (primitive type wrapper classes або value classes), оскільки їх екземпляри обгортають себе навколо значень примітивних типів.

- Java постачає 8 класів-обгорток примітивних типів з 2 причин:
  - Структури даних з Collections Framework (lists, sets, maps) можуть містити лише об'єкти, але не значення примітивних типів. Такі значення зберігаються в екземплярах класів-обгорток.
  - Обгортки – зручне місце для опису пов'язаних із значенням констант, наприклад, MAX_VALUE та MIN_VALUE) та методів класу, зокрема методи Integer's parseInt() та Character's isDigit(), isLetter(), toUpperCase()) з примітивними типами.

Клас Boolean

- Boolean – найменший з класів-обгорток для примітивних типів.
  - Оголошує 3 константи, включаючи TRUE і FALSE, які описують заздалегідь створені Boolean-об’єкти.
- Також оголошує пару конструкторів для ініціалізації об’єкту Boolean:
  - Boolean(boolean value)
  - Boolean(String s) конвертує текст у значення true або false і зберігає це значення в об’єкті Boolean.
- Другий конструктор порівнює s та true.
  - Порівняння нечутливе до регістра: можна передати true, TRUE, tRue тощо, щоб зберегти значення true в об’єкті.
  - Інакше конструктор зберігає значення false.

- `int compareTo(Boolean b)` compares the current `Boolean` object with `b` to determine their relative order. The method returns 0 when the current object contains the same Boolean value as `b`, a positive value when the current object contains true and `b` contains false, and a negative value when the current object contains false and `b` contains true.
- `boolean equals(Object o)` compares the current `Boolean` object with `o` and returns true when `o` is not null, `o` is of type `Boolean`, and both objects contain the same Boolean value.
- `static boolean getBoolean(String name)` returns true when a system property (discussed later in this chapter) identified by `name` exists and is equal to true.
- `int hashCode()` returns a suitable hash code that allows `Boolean` objects to be used with hash-based collections (discussed in Chapter 9).
- `static boolean parseBoolean(String s)` parses `s`, returning true when `s` equals `"true"`, `"TRUE"`, `"True"`, or any other uppercase/lowercase combination. Otherwise, this method returns false. (*Parsing* breaks a sequence of characters into meaningful components, known as *tokens*.)
- `String toString()` returns `"true"` when the current `Boolean` instance contains true; otherwise, this method returns `"false"`.
- `static String toString(boolean b)` returns `"true"` when `b` contains true; otherwise, this method returns `"false"`.
- `static Boolean valueOf(boolean b)` returns `TRUE` when `b` contains true or `FALSE` when `b` contains false.
- `static Boolean valueOf(String s)` returns `TRUE` when `s` equals `"true"`, `"TRUE"`, `"True"`, or any other uppercase/lowercase combination. Otherwise, this method returns `FALSE`.

- Часто краще використовувати TRUE або FALSE, ніж створювати Boolean-об'єкти.
  - Розглянемо метод, що повертає Boolean-об'єкт, що містить true, коли його аргумент типу double від'ємний, і false, якщо додатний.

```
Boolean isNegative(double d)
{
   return new Boolean(d < 0);
}
```

  - Створювати окремий об'єкт не обов'язково. При частому виклику методу створюється багато Boolean-об'єктів, що займають пам'ять кучі.
  - Збирач сміття при нестачі кучі сповільнює додаток.

```
Boolean isNegative(double d)
{
   return (d < 0) ? Boolean.TRUE : Boolean.FALSE;
}
```

Tip You should strive to create as few objects as possible. Not only will your applications have smaller memory footprints, they'll perform better because the garbage collector will not run as often.

Клас Character

- Найбільший клас-обгортка, що містить багато констант, конструктор, методи та кілька вкладених класів (Subset і UnicodeBlock).
- Character оголошує єдиний конструктор Character(char value), який використовується для ініціалізації об’єкту Character значенням.
  - Конструктор доповнюється методом char charValue(), що повертає обгорнуте значення.
- Часто в коді записують вирази, подібні до ch >= '0' && ch <= '9‘ або ch >= 'A' && ch <= 'Z'. Цього краще уникати:
  - Дуже просто «зловити баг». Наприклад, написавши ch > '0' && ch <= '9'.
  - Вирази не дуже змістовні для розуміння того, що вони тестують.
  - Вирази мають ухил до арабських цифр (0–9) та латинських букв (A–Z, a–z).
  - Вони не враховують digits and letters that are valid in other languages. Наприклад, '\u0beb' – символьний літерал, що представляє одну з цифр у Tamil language.

Методи класу Character

- Оголошуються кілька методів класу для порівняння та конвертації:
  - **`static boolean isDigit(char ch)`** повертає true, коли ch містить цифру (зазвичай від 0 до 9, але і цифри з інших алфавітів).
  - **`static boolean isLetter(char ch)`** повертає true, коли ch містить букву.
  - **`static boolean isLetterOrDigit(char ch)`** повертає true, коли ch містить букву або цифру.
  - **`static boolean isLowerCase(char ch)`** повертає true, коли ch містить маленьку літеру.
  - **`static boolean isUpperCase(char ch)`** повертає true, коли ch містить велику літеру.
  - **`static boolean isWhitespace(char ch)`** повертає true, коли ch містить whitespace character (пробіл, табуляцію, повернення каретки або line feed).
  - **`static char toLowerCase(char ch)`** повертає lowercase-еквівалент великої букви ch; інакше повертає значення ch.
  - **`static char toUpperCase(char ch)`** повертає uppercase-еквівалент маленької букви ch; інакше повертає значення ch.
- Наприклад, краще віддавати перевагу isDigit(ch), а не ch >= '0' && ch <= '9'.

Класи Float і Double

- Зберігають дробові значення одинарної та подвійної точності в об’єктах Float і Double.
- Константи
  - **`MAX_VALUE`** – максимальне значення, яке мають типи float або double.
  - **`MIN_VALUE`** – мінімальне значення, яке мають типи float або double.
  - **`NaN`** – результат 0.0F/0.0F для float і 0.0/0.0 як double.
  - **`NEGATIVE_INFINITY`** представляє −∞ для float або double.
  - **`POSITIVE_INFINITY`** представляє +∞ для float або double.
- Конструктори
  - **`Float(float value)`** ініціалізує Float-об’єкт значенням value.
  - **`Float(double value)`** initializes the Float object to the float equivalent of value.
  - **`Float(String s)`** converts s’s text to a floating-point value and stores this value in the Float object.
  - **`Double(double value)`** ініціалізує Double-об’єкт значенням value.
  - **`Double(String s)`** перетворює текст змінної s у дробове значення подвійної точності та зберігає це значення в об’єкті Double.

Методи класів

- Float
  - **`static int floatToIntBits(float value)`** конвертує значення в 32-бітове ціле.
  - **`static boolean isInfinite(float f)`** повертає true, коли значення f буде +infinity або -infinity. Пов'язаний з ним метод **`boolean isInfinite()`** повертає true, коли значення поточного Float-об'єкта дорівнює +infinity або -infinity.
  - **`static boolean isNaN(float f)`** повертає true, коли значення f дорівнює NaN. Пов'язаний з ним метод **`boolean isNaN()`** повертає true, коли значення поточного Float-об'єкта дорівнює NaN.
  - **`static float parseFloat(String s)`** виконує розбір рядка s, повертаючи дробове число, записане в текстововму представленні. Викидає NumberFormatException, коли представлення некоректне.
- Double
  - **`static long doubleToLongBits(double value)`** перетворює значення в довге ціле.
  - **`static boolean isInfinite(double d)`** повертає true, коли значення d дорівнює +infinity або -infinity. Пов'язаний з ним метод **`boolean isInfinite()`** повертає true, коли таке значення поточного Double-об'єкта.
  - **`static boolean isNaN(double d)`** returns true when d's value is NaN. A related public boolean isNaN() method returns true when the current Double object's value is NaN.
  - **`static double parseDouble(String s)`** parses s, returning the double precision floating-point equivalent of s's textual representation of a double precision floatingpoint value or throwing NumberFormatException when this representation is invalid.

Методи floatToIntBits() та doubleToIntBits()

- Методи floatToIntBits() та doubleToIntBits() використовуються при реалізації методів equals() і hashCode(), що повинні брати до уваги поля типів float і double відповідно.

- Вони дозволяють цим методам коректно обробляти наступні випадки:
  - equals() має повертати true, коли f1 та f2 мають значення Float.NaN (аналогсічно з Double.NaN). Якщо equals() було реалізовано на зразок f1.floatValue == f2.floatValue(), цей метод поверне false, оскільки NaN нічому не дорівнює, включаючи самого себе.
  - equals() має повертати false тоді, коли f1 = +0.0, f2 = -0.0 (і навпаки). Для аналогічної до першого пункту реалізації метод поверне true (+0.0 == -0.0)

```
public class FloatDoubleDemo
{
   public static void main(String[] args)
   {
      Float f1 = new Float(Float.NaN);
      System.out.println(f1.floatValue());
      Float f2 = new Float(Float.NaN);
      System.out.println(f2.floatValue());
      System.out.println(f1.equals(f2));
      System.out.println(Float.NaN == Float.NaN);
      System.out.println();
      Double d1 = new Double(+0.0);
      System.out.println(d1.doubleValue());
      Double d2 = new Double(-0.0);
      System.out.println(d2.doubleValue());
      System.out.println(d1.equals(d2));
      System.out.println(+0.0 == -0.0);
   }
}
```

- Вивід

```
NaN
NaN
true
false
0.0
-0.0
false
true
```

- Коли бажаєте протестувати float або double-значення на рівність з +infinity або -infinity (проте не обох), не використовуйте isInfinite().
  - Замість цього порівнюйте значення з NEGATIVE_INFINITY або POSITIVE_INFINITY за допомогою ==.
  - Наприклад, f == Float.NEGATIVE_INFINITY.

Integer, Long, Short, and Byte

- Зберігають 32-bit, 64-bit, 16–bit, and 8-bit integer values в об'єктах Integer, Long, Short, and Byte відповідно.
- Константи: MAX_VALUE та MIN_VALUE визначають максимальне та мінімальне значення примітивних типів.
- Конструктори
  - **`Integer(int value)`** ініціалізує Integer-об'єкт значенням.
  - **`Integer(String s)`** конвертує текст з рядка s в 32-бітне ціле значення та зберігає його в об'єкті Integer.
  - **`Long(long value)`** ініціалізує значенням Long-об'єкт.
  - **`Long(String s)`** конвертує текст з рядка s в 64-бітне ціле значення та зберігає його в об'єкті Long.
  - **`Short(short value)`** ініціалізує значенням Short-об'єкт.
  - **`Short(String s)`** converts s's text to a 16–bit integer value and stores this value in the Short object.
  - **`Byte(byte value)`** ініціалізує Byte-об'єкт значенням value.
  - **`Byte(String s)`** конвертує текст з s у 8-тове ціле число та зберігає його в Byte-об'єкті.

Методи

- static String toBinaryString(int i) повертає об’єкт String, що містить бінарне представлення числа i.
  - Наприклад, Integer.toBinaryString(255) повертає об’єкт, що містить 11111111.
- static String toHexString(int i) returns a String object containing i’s hexadecimal representation.
  - For example, Integer.toHexString(255) returns a String object containing ff.
- static String toOctalString(int i) returns a String object containing i’s octal representation.
  - For example, toOctalString(64) returns a String object containing 100.
- static String toString(int i) returns a String object containing i’s decimal representation.
  - For example, toString(255) returns a String object containing 255.
- Часто зручно дописувати нулі перед бінарним рядком, щоб вирівнювати кілька таких рядків в стовпчику.
  - Наприклад, бажаєте створити додаток, що виводить:

```
11110001
+
00000111
--------
11111000
```

Вирівнювання бінарних рядків

```
public class AlignBinaryString
{
   public static void main(String[] args)
   {
      System.out.println(toAlignedBinaryString(7, 8));
      System.out.println(toAlignedBinaryString(255, 16));
      System.out.println(toAlignedBinaryString(255, 7));
   }

   static String toAlignedBinaryString(int i, int numBits)
   {
      String result = Integer.toBinaryString(i);
      if (result.length() > numBits)
         return null; // cannot fit result into numBits columns
      int numLeadingZeros = numBits - result.length();
      StringBuilder sb = new StringBuilder();
      for (int j = 0; j < numLeadingZeros; j++)
         sb.append('0');
      return sb.toString() + result;
   }
}
```

- Метод toAlignedBinaryString() приймає 2 аргументи:
  - 32-бітне ціле, що конвертуватиметься в бінарний рядок
  - Кількість бітових стовпців in which to fit the string.

Вивід:

```
00000111
0000000011111111
null
```

- Після виклику toBinaryString(), що повертає бінарний еквівалент і без значущих нулів, toAlignedBinaryString() забезпечує відповідність розрядів числа та кількості стовпчиків numBits.
- Далі toAlignedBinaryString() обчислює кількість leading "0", що передують результату, а потім використовує цикл for, щоб створити рядок of leading zeros.
  - Метод закінчує поверненням leading zeros string prepended to the result string.

ДЯКУЮ ЗА УВАГУ!

Наступне запитання: рядки та масиви як посилальні типи даних