

Java Core

Заняття 1

Java - мова, яка допускає довільне форматування тексту програм. Для того, щоб програма працювала нормальн, немає ніякої необхідності вирівнювати її текст спеціальним чином.

Коментарі

Хоча коментарі ніяк не впливають на виконуваний код програми, при правильному використанні вони виявляються дуже істотною частиною вихідного тексту. Існує три різновиди коментарів: [коментарі в одному рядку](#), [коментарі в декількох рядках](#) і, нарешті, [коментарі для документування](#).

Коментарі, що займають один рядок, починаються з символів `//` і закінчуються в кінці рядка. Такий стиль коментування корисний для розміщення коротких пояснень до окремих рядків коду:

```
a = 42; // Якщо 42 - відповідь, то яке ж було запитання?
```

Для більш докладних пояснень ви можете скористатися коментарями, розміщеними на декількох рядках, почавши текст коментарів символами `/*` і закінчивши символами `*/`. При цьому весь текст між цими парами символів буде розрінений як коментар і транслятор його проігнорує.

```
/*
Цей код дещо хитромудрий ...
Спробую пояснити:
...
*/
```

Третя, особлива форма коментарів, призначена для сервісної програми `javadoc`, яка використовує компоненти Java-транслятора для автоматичної генерації документації по інтерфейсах класів. Правило, що використовується для коментарів цього виду, наступне: для того, щоб розмістити перед оголошенням відкритого (`public`) класу, методу або змінної що документує коментар, потрібно почати його з символів `/**` (коса риска і дві зірочки). Закінчується такий коментар точно так само, як і звичайний коментар - символами `*/`. Програма `javadoc` вміє розрізняти в документуючих коментарях деякі спеціальні змінні, імена яких починаються з символу `@`. Ось приклад такого коментаря:

```
 /**
 * Цей клас вміє робити чудові речі. Радимо кожному, хто
 * захоче написати ще більш досконалій клас, взяти його в якості
 * базового.
 * @see java.applet.Applet
 * @author Firstname Secondname
 * @version 1.2
 */
class CoolApplet extends Applet { ... }

 /**
 * У цього методу два параметри:
 * param key - це ім'я параметра.
 * param value - це значення параметра з ім'ям key.
 */
void put (String key, Object value) { ... }
```

Зарезервовані ключові слова

Зарезервовані ключові слова - це спеціальні ідентифікатори, які в мові Java використовуються для того, щоб ідентифікувати вбудовані типи, модифікатори і засоби управління виконанням програми. На сьогоднішній день в мові Java є 59 зарезервованих слів (див. Таблицю 1). Ці ключові слова спільно з синтаксисом операторів і роздільників входять в опис мови Java. Вони можуть застосовуватися тільки за призначенням, їх не можна використовувати в якості ідентифікаторів для імен змінних, класів або методів.

Таблиця 1

Зарезервовані слова Java

abstract	boolean	break	byte	byvalue
case	cast	catch	char	class
const	continue	default	do	double
else	extends	false	final	finally
float	for	future	generic	goto
if	implements	import	inner	instanceof
int	interface	long	native	new
null	operator	outer	package	private
protected	public	rest	return	short
static	super	switch	synchronized	this
throw	throws	transient	true	try
var	void	volatile	while	

Відзначимо, що слова `by value`, `cast`, `const`, `future`, `generic`, `goto`, `inner`, `operator`, `outer`, `rest`, `var` зарезервовані в Java, але поки не використовуються. Крім цього, в Java є зарезервовані імена методів (ці методи успадковуються кожним класом, їх не можна використовувати, за винятком випадків явного перевизначення методів класу `Object`).

Таблиця 2

Зарезервовані імена методів Java

clone	equals	finalize	getClass	hashCode
notify	notifyAll	toString	wait	

Ідентифікатори

Ідентифікатори використовуються для іменування класів, методів і змінних. В якості ідентифікатора може використовуватися будь-яка послідовність малих і великих літер, цифр і символів `_` (підкреслення) і `$` (долар). Ідентифікатори не повинні починатися з цифри, щоб транслятор не переплутав, їх з числовими літеральними константами, які будуть описані нижче. Java - мова, чутлива до регістру букв. Це означає, що, наприклад, `Value` і `VALUE` - різні ідентифікатори.

Літерали

Значення в Java задаються їх літеральним поданням. Цілі числа, числа з плаваючою точкою, логічні значення, символи і рядки можна розташовувати в будь-якому місці вихідного коду. Типи будуть розглянуті далі.

Цілі літерали

[Цілі числа](#) - це тип, який використовується в звичайних програмах найбільш часто. Будь-яке ціличисельне значення, наприклад, 1, 2, 3, 42 - це цілій літерал. У даному прикладі наведено десяткові числа, тобто числа з основою 10 - саме ті, які ми повсякденно використовуємо поза світу комп'ютерів. Крім десяткових, як цілих літералів можуть використовуватися також числа з основою 8 і 16 - вісімкові і шістнадцяткові. Java розпізнає [вісімкові числа](#) по стоячому попереду нулю. Нормальний десяткові числа не можуть починатися з нуля, так що використання в програмі зовні допустимого числа 09 приведе до повідомлення про помилку при трансляції, оскільки 9 не входить в діапазон 0 .. 7, допустимий для знаків вісімкового числа. [Шістнадцяткове](#) значення розрізняється по стоячим попереду символам нуль-х (0x або 0X). Діапазон значень шістнадцяткової цифри - 0 .. 15, причому в якості цифр для значень 10 .. 15 використовуються букви від A до F (або від a до f). За допомогою шістнадцяткових чисел ви можете в короткою і ясною формами представити значення, орієнтовані на використання в комп'ютері, наприклад, написавши 0xffff замість 65535.

Цілі літерали є значеннями типу [int](#), яке в Java зберігається в 32-бітовому слові. Якщо вам потрібне значення, яке по модулю більше, ніж приблизно 2 мільярди, необхідно скористатися константою типу [long](#). При цьому число буде зберігатися в 64-бітовому слові. До числа з будь-яким з названих вище основ ви можете приписати праворуч стрічкову або прописну букву L, вказавши таким чином, що дане число відноситься до типу long. Наприклад, 0x7fffffffffffffL або 9223372036854775807L - це значення, найбільше для числа типу long.

Літерали з плаваючою точкою

Числа з плаваючою точкою представляють десяткові значення, у яких є дрібна частина. Їх можна записувати або в звичайному, або експоненціальному форматах. У звичайному форматі число складається з деякої кількості десяткових цифр, що стоїть після них десяткового дробу, і наступних за ним десяткових цифр дробової частини. Наприклад, 2.0, 3.14159 і .6667 - це допустимі значення чисел з плаваючою точкою, записаних у стандартному форматі. У експоненціальному форматі після перерахованих елементів додатково зазначається десятковий порядок. Порядок визначається позитивним або негативним десятковим числом, наступним за символом E або e. Приклади чисел в експоненціальному форматі: 6.022e23, 314159E-05, 2e + 100. У Java числа з плаваючою точкою за замовчуванням розглядаються, як значення типу double. Якщо вам потрібна константа типу float, праворуч до літералу треба приписати символ F або f. Якщо ви любитель надлишкових визначень - можете додавати до літералу типу double символ D або d. Значення використовуваного за замовчуванням типу double зберігаються в 64-бітовому значенні, менш точні значення типу float - в 32-бітових.

Логічні літерали

У логічної змінної може бути лише два значення - true (істина) і false (неправда). Логічні значення true і false НЕ перетворюються ні в яке числове подання. Ключове слово true в Java не

дорівнює 1, а false не дорівнює 0. У Java ці значення можуть присвоюватися тільки змінним типу boolean або використовуватися у виразах з логічними операторами.

Символьні літерали

Символи в Java - це індекси в таблиці символів UNICODE. Вони являють собою 16-бітові значення, які можна перетворити в цілі числа і до яких можна застосовувати оператори цілочисельної арифметики, наприклад, оператори додавання і віднімання. Символьні літерали поміщаються всередині пари апострофів (' '). Всі видимі символи таблиці ASCII можна прямо вставляти всередину пари апострофів: - 'a', 'z', '@'. Для символів, які неможливо ввести безпосередньо, передбачено кілька керуючих послідовностей.

Керуючі послідовності символів

Управляющая последовательность	Описание
\ddd	Восьмеричный символ (ddd)
\xxxxx	Шестнадцатиричный символ UNICODE (xxxx)
'	Апостроф
"	Кавычка
\\\	Обратная косая черта
\r	Возврат каретки (carriage return)
\n	Перевод строки (line feed, new line)
\f	Перевод страницы (form feed)
\t	Горизонтальная табуляция (tab)
\b	Возврат на шаг (backspace)

Рядкові літерали

Рядкові літерали в Java виглядають точно так само, як і в багатьох інших мовах - це довільний текст, укладений в пару подвійних лапок (""). Хоча рядкові літерали в Java реалізовані вельми своєрідно (Java створює об'єкт для кожного рядка), зовні це ніяк не проявляється. Приклади рядкових літералів: "Hello World!"; "Два \n рядка"; "\" А це в лапках \"". Всі керуючі послідовності і вісімкові/шістнадцяткові форми записів, які визначені для символьних літералів, працюють точно так само і в рядках. Рядкові літерали в Java повинні починатися і закінчуватися в одному і тому ж рядку вихідного коду. У цій мові, на відміну від багатьох інших, немає керуючої послідовності для продовження рядкового літерала на новому рядку.

Оператори

Оператор - це щось, що виконує деяку дію над одним або двома аргументами і видає результат. Синтаксично оператори найчастіше розміщуються між ідентифікаторами і літералами. Детально оператори будуть розглянуті їх перелік наведено в таблиці нижче.

+	+=	-	-=
*	*=	/	/=
	=	^	^=
&	&=	%	%=
>	>=	<	<=
!	!=	++	--
>>	>>=	<<	<<=
>>>	>>>=	&&	
==	=	~	?:
	instanceof	[]	

Розділювачі

Лише кілька груп символів, які можуть з'являтися в синтаксично правильної Java-програмі, все ще залишилися неназваним. Це - прості роздільники, які впливають на зовнішній вигляд і функціональність програмного коду.

Символы	Название	Для чего применяются
()	круглые скобки	Выделяют списки параметров в объявлении и вызове метода, также используются для задания приоритета операций в выражениях, выделения выражений в операторах управления выполнением программы, и в операторах приведения типов.
{ }	фигурные скобки	Содержат значения автоматически инициализируемых массивов, также используются для ограничения блока кода в классах, методах и локальных областях видимости.
[]	квадратные скобки	Используются в объявлениях массивов и при доступе к отдельным элементам массива.
;	точка с запятой	Разделяет операторы.
,	запятая	Разделяет идентификаторы в объявлениях переменных, также используется для связи операторов в заголовке цикла for.
.	точка	Отделяет имена пакетов от имен подпакетов и классов, также используется для отделения имени переменной или метода от имени переменной.

Змінні

Змінна - це основний елемент зберігання інформації в Java-програмі. Змінна характеризується комбінацією ідентифікатора, типу та області дії. Залежно від того, де ви оголосили змінну, вона може бути локальною, наприклад, для коду всередині циклу for, або це

може бути змінна екземпляра класу, доступна всім методам даного класу. Локальні області дії оголошуються за допомогою фігурних дужок.

Оголошення змінної

Основна форма оголошення змінної така:

тип ідентифікатор [= значення] [, ідентифікатор [= значення 2 ...];

Тип - це або один з вбудованих типів, тобто, `byte`, `short`, `int`, `long`, `char`, `float`, `double`, `boolean`, або ім'я класу або інтерфейсу. Нижче наведено кілька прикладів оголошення змінних різних типів. Зверніть увагу на те, що деякі приклади включають в себе ініціалізацію початкового значення. Змінні, для яких початкові значення не вказані, автоматично ініціалізуються нулем.

<code>int a, b, c;</code>	Объявляет три целых переменных a, b, c.
<code>int d = 3, e, f = 5;</code>	Объявляет еще три целых переменных, инициализирует d и f.
<code>byte z = 22;</code>	Инициализирует z.
<code>double pi = 3. 14159;</code>	Объявляет число пи (не очень точное, но все таки пи).
<code>char x = 'x';</code>	Переменная x получает значение 'x'.

У наведеному нижче прикладі створюються три змінні, відповідні сторонам прямокутного трикутника, а потім за допомоги теореми Піфагора обчислюється довжина гіпотенузи, в даному випадку числа 5, величини гіпотенузи класичного прямокутного трикутника зі сторонами 3-4-5.

```
class Variables {
    public static void main (String args []) {
        double a = 3;
        double b = 4;
        double c;
        c = Math.sqrt (a * a + b * b);
        System.out.println ("c =" + c);
    }
}
```

Домашня робота:

1. Написати по одній змінній кожного типу примітивних типів даних.
2. Вивести на консоль мінімальне і максимальне значення типів даних. Вивід здійснюється за допомогою `System.out.println()`. Логічному типу задати значення `true`.

Заняття №2

1. Що таке клас?

Клас - це певний шаблон, опис не тільки самої структури даних (що там може або повинно бути), а й поведінки, функціональних можливостей об'єктів цього класу.

2. Що таке об'єкт?

Об'єкт класу - це змінна типу класу, яка містить конкретні дані і поводиться відповідно до описаної в класі функціональності.

3. Що таке поля?

У класі описуються властивості майбутніх об'єктів - поля, що містять дані, і є цими властивостями.

4. Що таке методи?

Методи - функції для будь-яких взаємодій (з цими даними, з іншими об'єктами, з оточенням, з операційною системою і т.д.).

```
Клас: Автомобіль {
    ...
    змінна марка: String; //поля стрічка
    змінна модель: String; //поле стрічка
    ...
    функція заголовокДляПрайсЛиста() { //Метод
        повернути марка + " " + модель;
    }
}

class Auto{
    String brend = "Audi ";
    String mark = "Q7 ";

    String priceList (){
        return mark + " " + brend;
    }
}
```

5. Що таке конструктор?

Конструктор – це метод класу призначений для створення об'єктів цього класу, та їх ініціалізації.

Конструктор схожий з методом, але відрізняється від методу тим, що **не має** явним чином певного типу **повертаючих даних**, що **не успадковується**. Конструктори виділяються наявністю однакового імені з ім'ям класу, в якому оголошується.

Види конструкторів:

a) за замовчуванням

```
class Person {
    Person(){/*насправді цього писати не потрібно, тут зображене яким чином виглядає */
    }      /*конструктор, який автоматично створюється та викликається за допомогою JVM */
}
```

Вище написаний опис класу є еквівалентний цьому

```
class Person {
```

б) з параметрами

```
class Person{

    String name;
    int age;

    Person(String name, int age){
        this.name = name;
        this.age = age;
    }
}
```

в) без параметрів

```
class Person{

    String name;
    int age;

    Person(){
        this.name = "Andrew";
        this.age = 21;
    }
}
```

Деякі відмінності між конструкторами та іншими методами Java:

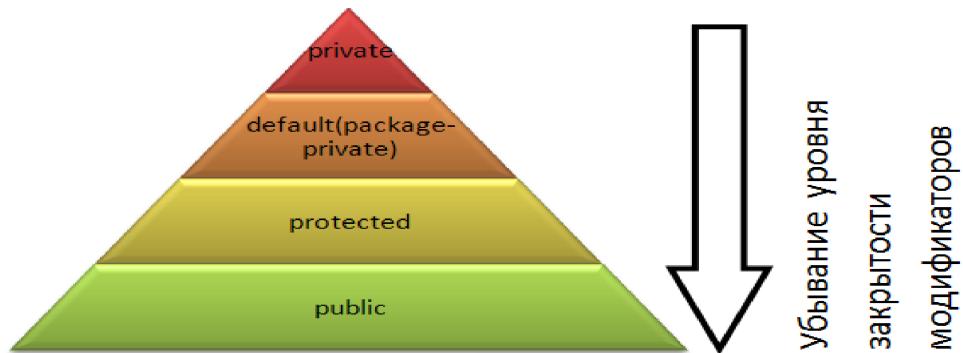
- конструктори не мають типу повертаючих даних (насправді вони завжди повертують this);
- конструктори не можуть безпосередньо викликатися (необхідно використовувати ключове слово new);
- конструктори не можуть мати модифікатори synchronized, final, abstract, native і static;

6. Модифікатори доступу?

- **private**: члени класу доступні тільки усередині класу;
- **default** (package-private) (модифікатор, по-замовчуванням): члени класу видно всередині пакету (якщо клас буде так оголошений, він буде доступний тільки всередині пакета);
- **protected**: члени класу доступні всередині пакету і для спадкоємців;
- **public**: члени клас доступні всім;

Послідовність модифікаторів спаданням рівня закритості: **private, default, protected, public**.

Під час спадкування можливо зміни модифікаторів доступу в бік більшої видимості. Так зроблено для того, щоб не порушувався принцип LSP для успадкованого класу.



Домашня робота : Написати клас Rectangle , в якому буде описано поля : довжина, ширина. Описати дані поля в конструкторі. Створити конструктор з параметрами і без параметрів. В конструкторі без параметрів описати свої дані. Тобто створюючи об'єкт, за вибіру даного конструктору, ви отримаєте прямокутник даного розміру. Написати методи, які будуть розраховувати площину та периметр. В Main класі продемонструвати створення об'єктів використовуючи два конструктори. Вивести на консоль площину і периметр прямокутника. Вивід на консолі повинен мати наступний вигляд: «Площа прямокутника = тут буде площа», «Периметр прямокутника = тут буде периметр» .

Заняття №3

1. Призначення Getters and Setters?

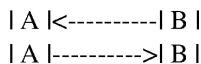
Основне призначення це отримання доступу до **private** полів.

```
class Person{
    private String name;
    private int age;
    Person(String name, int age){
        this.name=name;
        this.age=age;
    }
    public String getName(){ //гетер
        return this.name;
    }
    public void setName(String Name){ //сетер
        this.name=Name;
    }
    public int getAge(){ //гетер
        return this.age;
    }
    public void setAge(int Age){ //сетер
        this.age=Age;
    }
}
```

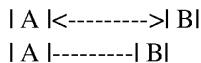
2. Асоціація, композиція і агрегація?

Асоціація - це найбільш загальний вид зв'язку між класами, який практично нічого конкретного про цей зв'язок не повідомляє. Асоціація відбувається, коли оголошується, що властивість класу містить посилання на екземпляр (екземпляри) іншого класу. Асоціації можуть бути одностороннimi і двостороннimi.

Приклади позначення односпрямованої асоціації:



Приклади позначення двобічної асоціації:



Стрілка вказує на той клас, від якого залежить інший клас. Так якщо в класі A є посилання на клас B, але не навпаки, то тоді цей зв'язок має позначатися так: | A |----->| B |. Композиція і Агрегація це часткові випадки Асоціації.

Агрегація, як і Композиція - це односпрямована асоціація, що має сенс Ціле-Частини. Ціле в цьому випадку має всередині себе посилання на частину, а частина не має всередині себе посилання на ціле. При цьому частина може в один момент часу знаходитися тільки в одному цілому. В агрегації обидва об'єкти можуть існувати незалежно: якщо контейнер буде знищений, то його вміст - ні.

Приклад:

```
public class Operation {
    int square(int R){
        return (int) (Math.PI*Math.pow(R,2));
    }
}

public class Circle {
    Operation op;//агрегація
    double pi=3.14;

    double area(int radius){
        op=new Operation(); /*агрегація -ми робимо об'єкт класу будь-де в класі, таким чином
        вони не об'єднані терміном існування */
        int rsquare=op.square(radius);
        return pi*rsquare;
    }
}

public class Main {
    public static void main(String[] args) {
        Circle c=new Circle();
        double result=c.area(5);
        System.out.println(result);
    }
}
```

Композиція - це агрегація, але коли включається об'єкт не може існувати без свого контейнера. Наприклад, факультет сам по собі може існувати тільки в складі якогось університету, і не може бути сам по собі.

Об'єкти в композиції не можуть існувати незалежно. Якщо контейнер буде знищений, то і всі його частини повинні бути знищені. При композиції сенсу в окремому існуванні будь-якої частини без його контейнера немає.

```
public class Operation {

    int square(int R){
        return (int) (Math.PI*Math.pow(R, 2));
    }

}

public class Circle {
    Operation op;// агрегація , ми створюємо посилання на інший клас
    double pi = 3.14;

    Circle() {
        op = new Operation(); // Композиція- ми обєднуюємо за терміном існування наші
    }

    double area(int radius) {

        int rSquare = op.square(radius);
        return pi * rSquare;
    }

}

public class Main {

    public static void main(String[] args) {

        Circle c=new Circle();
        double result=c.area(5);
        System.out.println(result);

    }

}
```

3. Наслідування – Це розширення можливостей одного класу іншим.

Тим самим ви отримуєте доступ до всіх полів і методів базового класу. Використовуючи наслідування, можна створити загальний клас, які визначає характеристики, загальні для набору пов'язаних елементів. Потім ви можете успадковуватися від нього і створити новий клас, який буде мати свої унікальні характеристики. Клас від якого наслідують в Java називають супер класом. Наслідуючий клас називають під класом.

Приклад:

```
public class Robot {

    public void work(){
        System.out.println("Я робот - я просто працюю");
    }
}

public class CoffeRobot extends Robot{

    @Override
```

```

public void work() {
    System.out.println("Я CoffeRobot - я варю каву");
}
}

```

Наслідування проводиться за допомогою ключового слова `extends`.

```

subClass extends superclass{
//some code...
}

```

4. Ключове слово super?

Якщо ваш метод перевизначає один з методів свого супер класу, ви можете викликати метод заміщення за рахунок використання ключового слова `super`. Ви також можете використовувати `super` ставляться до прихованої області (хоча ховається поля не рекомендується).

5. Клас Object?

Об'єкт це базовий клас для всіх інших об'єктів в Java. Кожен клас успадковується від `Object`. Відповідно всі класи успадковують методи класу `Object`.

1. `public final native Class getClass()`- отримаємо поля класу, інформацію.(Рефлексія)
2. `public final native int hashCode()`-Повертає hashCode об'єкту, використовується коли в нас є багато об'єктів і нам потрібна швидка вибірка об'єкту, то по hashCode це є найшвидше .
3. `public boolean equals(Object obj)`-порівнює два об'єкти, якщо вони рівні то повертає true в іншому випадку false.
4. `protected native Object clone() throws CloneNotSupportedException`- повертає клон об'єкта.
5. `public String toString()`-повертає стрічку, перетворює об'єкт в стрічку.
6. `public final native void notify()`-пробуджує потік після `wait()`, решта які очікують, будуть далі очікувати
7. `public final native void notifyAll()`-пробуджує всі потоки які очікують по `wait()`-присипляє потік
8. `public final native void wait(long timeout) throws InterruptedException`-присипляє потік на визначений термін
9. `public final void wait() throws InterruptedException`- присипляє потік на визначений термін
10. `protected void finalize() throws Throwable`- Що має зробити об'єкт перед тим як його збирач сміття знищить

6. Override vs Overload?

Override-Перевизначення використовується тоді, коли ви переписуєте (переробляєте, перевизначаєтеся) ВЖЕ існуючий метод.

Приклад :

```

public class Robot {

    public void work(){
        System.out.println("Я робот - я просто працюю");
    }
}

public class CoffeRobot extends Robot{

```

```

@Override
public void work() { // перевизначення методу work() у класі CoffeRobot при
наслідуванні
    System.out.println("Я CoffeRobot - я варю каву");
}
}

```

Overload- ви створюєте метод з таким же ім'ям, але з іншим набором вхідних параметрів.

Приклад :

```

public class Robot {

    public void work(){
        System.out.println("Я робот - я просто працюю");
    }

    public void work(String arg){// перегрузка методу work()
        System.out.println(arg);
    }

}

```

Домашня робота:

Створити клас Robot від якого слід унаслідувати CoffeRobot, RobotDancer, RobotCoocker. В Robot є метод work() , в якому слід описати роботу яку виконує кожен з роботів.

- Для Robot в методі work() слід написати код, щоб на консоль виводився наступний текст «Я Robot – я просто працюю».
- Для CoffeRobot в методі work() слід написати код, щоб на консоль виводився наступний текст «Я CoffeRobot – я варю каву».
- Для RobotDancer в методі work() слід написати код, щоб на консоль виводився наступний текст «Я RobotDancer – я просто танцюю».
- Для RobotCoocker в методі work() слід написати код, щоб на консоль виводився наступний текст «Я RobotCoocker – я просто готую».

1. Створити в Main класі екземпляри вищеописаних класів і викликати до кожного з них метод work().
2. Створити в Main класі масив класу Robot , заповнити масив екземплярами вищеописаних класів. Створити цикл for , пройтись по всіх елементах масиву і викликати для кожного елементу масиву метод work().

Заняття №4

1.Ключове слово static?

Це модифікатор, який дозволяє викликати поля і методи , помічені цим модифікатором, не прив'язуючи їх до певного об'єкту, тобто викликати без створення об'єкту, а через клас.

Застосовується змінними, методами і навіть до певних фрагментів коду, які не є частиною методу.

Застосовується до внутрішніх класам, методам, змінним і логічним блокам

- Статичні змінні ініціалізуються під час завантаження класу
- Статичні змінні єдині для всіх об'єктів класу (однакова посилання)
- Статичні методи мають доступ тільки до статичних змінних
- До статичних методів і змінним можна звертатися через ім'я класу
- Статичні блоки виконуються під час завантаження класу
- Чи не static методи не можуть бути перевизначені як static
- Локальні змінні не можуть бути оголошенні як static
- Абстрактні методи не можуть бути static
- Static поля НЕ серіалізуються (тільки при реалізації інтерфейсу Serializable)
- Тільки static змінні класу можуть бути передані в конструктор з параметрами, викликає через слово super (// параметр //) або this (// параметр //)

2. Модифікатор final?

Модифікатор 'final' використовується до класів, методів і змінних. Поля final не можуть бути змінені.

Клас, оголошений з модифікатором 'final' не може мати підкласів. Таким чином модифікатор 'final' додається до класу, коли його специфікація заморожена. **Метод** не перевизначається. **Поле** не перезаписується.

3. Абстракція ?

Це спосіб виділення набору значущих характеристик об'єкта, відкинувши з розгляду не значущі.

4. Абстрактний клас?

Це клас в якого є хоч один абстрактний метод. На основі якого не можуть створюватися об'єкти. При цьому спадкоємці класу можуть бути не абстрактними, на їх основі об'єкти створювати, відповідно, можна.

Для того, щоб перетворити клас у абстрактний перед його ім'ям треба вказати модифікатор abstract.

```
public abstract class Person {
    public abstract void work();
}
```

Домашня робота:

Написати клас абстрактний Pet , в якому описати абстрактний метод voice(). Створити класи Cow, Cat, Dog , які наслідуються від Pet. Перевизначити в них метод voice(), так, щоб викликаючи його в методі main, на консоль виводилося : “Я кіт- Мяуу-Мяуу”, “Я пес- Гауу-Гауу”, “Я корова-Мууу-Мууу”.

Заняття №5

1. Поліморфізм?

Це можливість об'єктів із однаковою специфікацією, мати різну реалізацію. Найпростіше проявлення поліморфізму – це перевизначення методів при наслідуванні класів. [Поліморфізм](#) - це адаптація коду під багато форм.

2. Override i Overload?

Перевизначення і перегрузка методів.

Override-Перевизначення використовується тоді, коли ви переписуєте (переробляєте, перевизначаєте) ВЖЕ існуючий метод.

Приклад :

```
public class Robot {

    public void work(){
        System.out.println("Я робот - я просто працюю");
    }
}

public class CoffeRobot extends Robot{

    @Override
    public void work() { // перевизначення методу work() у класі CoffeRobot при
наслідуванні

        System.out.println("Я CoffeRobot - я варю каву");
    }
}
```

Overload- ви створюєте метод з таким же ім'ям, але з іншим набором вхідних параметрів.

Приклад :

```
public class Robot {

    public void work(){
        System.out.println("Я робот - я просто працюю");
    }

    public void work(String arg){ // перегрузка методу work()
        System.out.println(arg);
    }

}
```

3. Інтерфейс, різниця між інтерфейсом та Абстрактними класом?

Ключове слово [interface](#) використовується для створення повністю абстрактних класів. Творець інтерфейсу визначає імена методів, списки аргументів і типи повертаються значень, але не тіла методів.

Наявність слова [interface](#) означає, що саме так повинні виглядати всі класи, які реалізують даний інтерфейс. Таким чином, будь-який код, який використовує конкретний інтерфейс, знає тільки те, які методи викликаються для цього інтерфейсу, але не більше того.

Інтерфейси можна використовувати для імпорту констант в декілька класів. Ви просто оголошуєте інтерфейс, який містить змінні з потрібними значеннями. При реалізації інтерфейсу в класі імена змінних будуть поміщені в область констант. Поля для констант стають відкритими і є статичними і кінцевими (модифікатори `public static final`). При цьому, якщо інтерфейс не буде

містити ніяких методів, то клас не буде нічого реалізовувати.Хоча даний підхід не рекомендують використовувати.

Інтерфейс може успадковуватися від іншого інтерфейсу через ключове слово `extends`.

Приклад:

```
interface Monster {
    void eat();
}
```

Різниця між інтерфейсом та абстрактним класом

Різниця: Абстрактний (`abstract`) клас - клас, на основі якого не можуть створюватися обєкти; позначається як `abstract`.

Інтерфейс - такий же абстрактний клас, тільки в ньому не може бути властивостей і не визначені тіла у методів.

Так само варто зауважити, що абстрактний клас успадковується (`extends`), а інтерфейс реалізується (`implements`). От і виникає різниця між ними, що успадковувати ми можемо тільки 1 клас, а реалізувати скільки завгодно.

ВАЖЛИВО! При реалізації інтерфейсу, необхідно реалізувати всі його методи, інакше буде **Fatal error**, так само це можна уникнути, присвоївши слово `abstract`.

4. Різниця між extends vs implements ?

Extends один клас, implements безліч. Extends має зв'язок з класами **is a**, implements має зв'язок **has a**.

Домашня робота:

Створити interface Зарплата, в якому передбачити метод зарплата(). Створити клас Працівник з погодинною зарплатою, та Працівник з фіксованою місячною зарплатою . Реалізувати методи інтерфейсу в к класах : Працівник з погодинною зарплатою, та Працівник з фіксованою місячною зарплатою . Формула розрахунку зарплати працівника є довільною. Вивести на екран скільки заробляють перший і другий працівники.

Заняття 6 (Повторення ООП –Пишемо Тест)

1. Об'єкт як поле класу ?

Асоціація - це найбільш загальний вид зв'язку між класами, який практично нічого конкретного про цей зв'язок не повідомляє. Асоціація відбувається, коли оголошується, що властивість класу містить посилання на екземпляр (екземпляри) іншого класу. Асоціації можуть бути односторонніми і двосторонніми.

Приклади позначення односпрямованої асоціації:

| A |-----| B |

| A |----->| B |

Приклади позначення двобічної асоціації:

| A |----->| B |

| A |-----| B |

Стрілка вказує на той клас, від якого залежить інший клас. Так якщо в класі A є посилання на клас B, але не навпаки, то тоді цей зв'язок має позначатися так: | A |----->| B |. Композиція і Агрегація це часткові випадки Асоціації.

Агрегація , як і Композиція - це односпрямована асоціація, що має сенс Ціле-Частини. Ціле в цьому випадку має всередині себе посилання на частину, а частина не має всередині себе посилання на ціле. При цьому частина може в один момент часу знаходитися тільки в

одному цілому. В агрегації обидва об'єкти можуть існувати незалежно: якщо контейнер буде знищений, то його вміст - ні.

Композиція - це агрегація, але коли включається об'єкт не може існувати без свого контейнера. Наприклад, факультет сам по собі може існувати тільки в складі якогось університету, і не може бути сам по собі.

Об'єкти в композиції не можуть існувати незалежно. Якщо контейнер буде знищений, то і всі його частини повинні бути знищені. При композиції сенсу в окремому існуванні будь-якої частини без його контейнера немає.

2. Повторення Класи , об'єкти , перегрузка методів і наслідування?

Клас - це певний шаблон, опис не тільки самої структури даних (чого там може або повинно бути), а й поведінки, функціональних можливостей об'єктів цього класу.

Об'єкт класу - це змінна класу, яка містить конкретні дані і працює відповідно до описаної в класі функціональністю.

У класі описуються властивості майбутніх об'єктів - поля, що містять дані,

Методи - функції для будь-яких взаємодій (з цими даними, з іншими об'єктами, з оточенням, з операційною системою і т.д.).

Що таке конструктор?

Конструктор – це метод класу призначений для створення об'єктів цього класу, та ініціалізація їх.

Конструктор схожий з методом, але відрізняється від методу тим, що **не має явним чином певного типу повертається даних**, що **не успадковується**. Конструктори часто виділяються наявністю однакового імені з ім'ям класу, в якому оголошується.

Види конструкторів:

- за замовчуванням
- з параметрами
- без параметрів

Деякі відмінності між конструкторами та іншими методами Java:

- конструктори не мають типу повертається даних (насправді вони завжди повертають this);
- конструктори не можуть безпосередньо викликатися (необхідно використовувати ключове слово new);
- конструктори не можуть мати модифікатори synchronized, final, abstract, native і static;

Наслідування – Це розширення можливостей одного класу іншим.

Тим самим ви отримуєте доступ до всіх полів і методів базового класу.

Використовуючи спадкування, можна створити загальний клас, які визначає характеристики, загальні для набору пов'язаних елементів. Потім ви можете успадковуватися від нього і створити новий клас, який буде мати свої унікальні характеристики. Головний успадкований клас в Java називають супер класом. Наследуючий клас називають підкласом.

Ключове слово super?

Якщо ваш метод перевизначає один з методів свого супер класу, ви можете викликати метод заміщення за рахунок використання ключового слова супер. Ви також можете використовувати супер ставлятися до прихованої області (хоча ховається поле не рекомендується).

Домашня робота:

Створити клас Кермо, Колесо, Кузов - описати дані класи(getters, setters, toString). Всі поля повинні бути приватними. Додати методи які б виконували певні функції з полями, тобто збільшували розмір колеса у декілька разів, чи змінювали діаметр керма і т.д.. Створити клас Машина, який матиме деякі свої поля та об'єкти класу Кермо, Кузов, Колесо - як поля класу. Додати методи, які би змінювали стан полів класу , для виконання певних функцій, описати даний клас(getters, setters, toString). Всі поля повинні бути параметрами в конструкторі. В мейн методі , створити клас Машина, запустити всі його методи.

Заняття 7**1. Що таке Enum?**

Enums - це список іменованих констант. Оголошуючи enum ми неявно створюємо клас похідний від java.lang.Enum. Умовно конструкція enum Season {...} еквівалентна class Season extends java.lang.Enum {...}. І хоча явним чином успадковуватися від java.lang.Enum нам не дозволяє компілятор, все ж у тому, що enum успадковується, легко переконатися за допомогою reflection:

```
System.out.println(Season.class.getSuperclass());
```

На консоль буде виведено:

```
class java.lang.Enum
```

Власне наслідування за нас автоматично виконує компілятор Java. Далі давайте домовимося називати клас, створений компілятором для реалізації перерахування - enum-класом, а можливі значення перераховується типу - елементами enum-а.

Приклад :

```
public enum Season {
    WINTER, SPRING, SUMMER, AUTUMN;
```

Методи enumів:

1. **name()** - повертає ім'я константи , точно так само як і оголошено в enum класі.
2. **ordinal()**- повертає номер константи в enum класі. Рахунок починається від нуля
3. **toString()**- повертає ім'я константи , точно так само як і оголошено в enum класі.
4. **equals**- повертає true або false, порівнює enum з об'єктом.
5. **hashCode()**- повертає хешкод константи
6. **compareTo()**- порівнює дві константи одного enum класу. Повертає 0-якщо рівні, -1 і 1 якщо не рівні, це вже залежить від результату лексикографічного порівняння констант.
7. **getDeclaringClass()**- повертає адресу нашого enumа.
8. **valueOf()**- використовується для порівняння константи із стрічкою. Параметром в даний метод передається стрічка. якщо є така константа то повертає її ім'я.
9. **values**- повертає масив всіх констант класу.

10. Робота з класом Scanner?

Для введення даних використовується клас Scanner з бібліотеки пакетів Java.

Цей клас треба імпортувати в тій програмі, де він буде використовуватися. У класі є методи для читання чергового символу заданого типу зі стандартного потоку введення, а також для перевірки існування такого символу.

Для роботи з потоком введення необхідно створити об'єкт класу Scanner, при створенні вказавши, з яким потоком введення він буде пов'язаний. Стандартний потік вводу (клавіатура) в Java представлений об'єктом - `System.in`. А стандартний потік виводу (дисплей) - уже знайомим вам об'єктом `System.out`.

Метод `hasNextDouble()`, застосований до об'єкту класу Scanner, перевіряє, чи введене з потоку введення, дійсне число є типу double, а метод `nextDouble()` - читає його. Якщо спробувати вважати значення без попередньої перевірки, то під час виконання програми можна отримати помилку (відладчик заздалегідь таку помилку не виявить). Наприклад, спробуйте в представленій далі програмі ввести якесь дійсне число:

Є також метод `nextLine()`, що дозволяє зчитувати цілу послідовність символів, тобто рядок, а, значить, отримане через цей метод значення потрібно зберігати в об'єкті класу String. У наступному прикладі створюється два таких об'єкти, потім в них по черзі записується введення користувача, а далі на екран виводиться один рядок, отримана об'єднанням введених послідовностей символів.

Приклад:

```
public class Work {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);
        if(sc.hasNextDouble()){
            double d=sc.nextDouble();
        }
    }
}
```

11. Методи класу String ?

У класі String існує маса корисних методів, які можна застосовувати до рядків (перед ім'ям методу будемо вказувати тип того значення, яке він повертає)

- `int length()` - повертає довжину рядка (кількість символів в ній);
- `boolean isEmpty()` - перевіряє, порожня рядок;
- `String replace (a, b)` - повертає рядок, де символ a (літерал або змінна типу char) замінений на символ b;
- `String toLowerCase()` - повертає рядок, де всі символи вихідної рядки перетворені до рядковим;
- `String toUpperCase()` - повертає рядок, де всі символи вихідної рядки перетворені до прописних;
- `boolean equals (s)` - повертає істину, якщо рядок до якої застосований метод, збігається з рядком s зазначеної в аргументі методу (за допомогою оператора == рядки порівнювати не можна, як і будь-які інші об'єкти);
- `int indexOf (ch)` - повертає індекс символу ch в рядку (індекс це порядковий номер символу, але нумеруватися символи починають з нуля). Якщо символ зовсім не буде знайдений, то поверне -1. Якщо символ зустрічається в рядку несілько раз, то вовзвратіт індекс його первого входження.
- `int lastIndexOf (ch)` - аналогічний попередньому методу, але повертає індекс останнього входження, якщо смівол зустрівся в рядку кілька разів.
- `int indexOf (ch, n)` - повертає індекс символу ch в рядку, але починає перевірку з індексу n (індекс це порядковий номер символу, але нумеруватися символи починають з нуля).

- **char charAt (n)** - повертає код символу, що знаходиться в рядку під індексом n (індекс це порядковий номер символу, але нумеруватися символи починають з нуля).

Приклад:

```
public class A {
    public static void main(String[] args) {
        String s="Hello world";
        s.length();
        s.isEmpty();
    }
}
```

Домашня робота:

- 1) Написати консольну програму для роботи з Enums. Створити енум Сезони, в якому оголосити такі константи : Зима , Весна, Літо, Осінь. Створити енум Місяці, в якому створити 12 констант- місяці року(Січень , Лютий.. Грудень), оголосити змінну дні, та змінну сезон типу Сезон , як поле енума . Створити конструктор з визначеними параметрами в енумі Місяці, в який як параметри передати змінну дні та сезон. Описати getters до даних полів (дні, сезон). Створити консольне меню, в якому реалізувати наступні пункти :
 - Перевірити чи є такий місяць (*місяць вводимо з консолі, передбачити, щоб регистр букв був не важливим*)
 - Вивести всі місяці з такою ж порою року
 - Вивести всі місяці які мають таку саму кількість днів
 - Вивести на екран всі місяці які мають меншу кількість днів
 - Вивести на екран всі місяці які мають більшу кількість днів
 - Вивести на екран наступну пору року
 - Вивести на екран попередню пору року
 - Вивести на екран всі місяці які мають парну кількість днів
 - Вивести на екран всі місяці які мають непарну кількість днів
 - Вивести на екран чи введений з консолі місяць має парну кількість днів
- 2) Написати програму для роботи з Enums. Описати Енум Direction , в якому описати напрямки: North, South, West, East; Описати енумами материки світу(їх є 6). В константи материків вклсти константи частин світу , широту і довготу(double). Один материк описати одним довільним значенням широти і довготи. Створити мейн метод і попрацювати з методами класу енум (зі всіма) .
- 3) Створити програму, яка буде повідомляти, чи є ціле число, введене користувачем, парним або непарним. Якщо користувач введе не ціла число, то повідомляти йому про помилку.
- 4) Створити програму, яка буде обчислювати і виводити на екран суму двох цілих чисел, введених користувачем. Якщо користувач некоректно введе хоча б одне з чисел, то повідомляти про помилку.
- 5) Створити програму, яка буде перевіряти, чи є слово з п'яти букв, введене користувачем, паліндромом (приклади: «пилип», «ротор»). Якщо введено слово не з 5 букв, то повідомляти про помилку. Програма повинна нормально обробляти слово, навіть якщо в

ньому використані символи різного реєстра. Наприклад, слова «Комок» або «РОТОР» слід також вважати паліндромами.

Заняття 8

1). Collections framework?

Колекції представляють собою контейнер и для роботи з об'єктами.

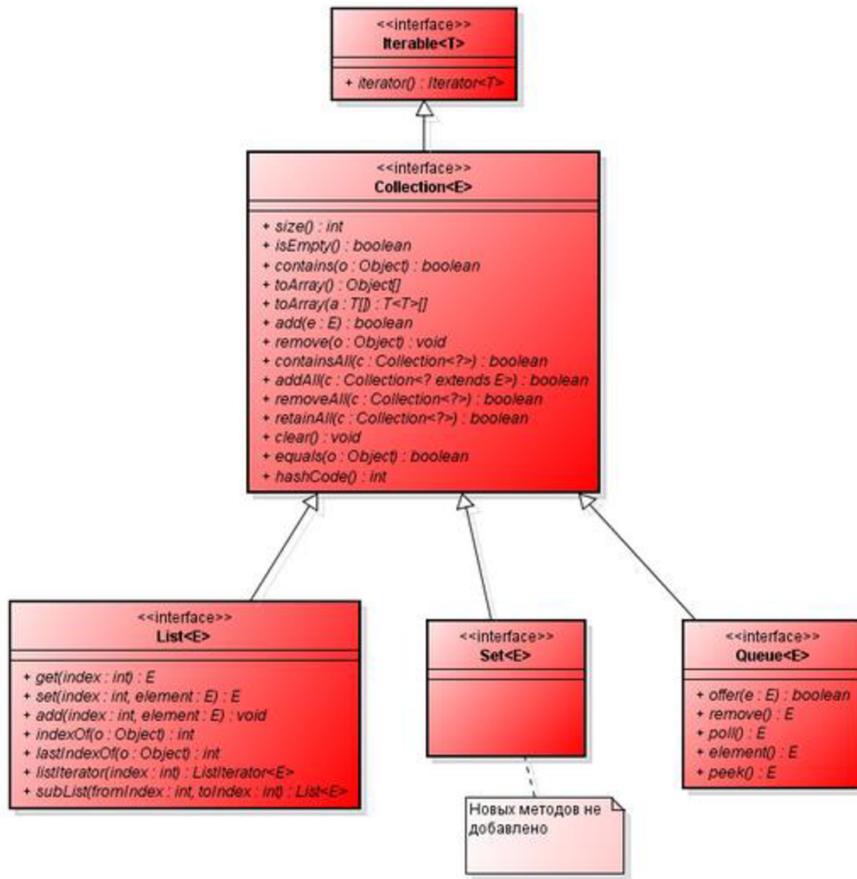
У бібліотеці колекцій Java існує два базових інтерфейсу, реалізації яких і являють сукупність всіх класів колекцій:

1. **Collection** - колекція містить набір об'єктів (елементів). Тут визначено основні методи для маніпуляції з даними, такі як вставка (add, addAll), видалення (remove, removeAll, clear), пошук (contains)

2. **Map** - описує колекцію, що складається з пар "ключ - значення". У кожного ключа тільки одне значення, що відповідає математичному поняттю однозначної функції або відображення (map). Таку колекцію часто називають ще словником (dictionary) або асоціативним масивом (associative array). Ніяк НЕ відноситься до інтерфейсу Collection і є самостійним.

Хоча фреймворк називається **Java Collections Framework**, інтерфейс мап та його реалізації входять в фреймворк теж!

Інтерфейси **Collection** і **Map** є базовими, але вони не є єдиними. Їх розширяють інші інтерфейси, що додають додатковий функціонал. Про них ми ще поговоримо.



2). Класифікація List, Set, Queue?

List - Являє собою неупорядковану колекцію, в якій допустимі дублюючі значення. Іноді їх називають послідовностями (sequence). Елементи такої колекції пронумеровані, починаючи від нуля, до них можна звернутися за індексом.

Set - описує неупорядковану колекцію, що не містить повторюваних елементів. Це відповідає математичному поняттю множини (set).

Queue - черга. Відразу запам'ятуємо як правильно вимовляється: Queue - КЮ (http://www.youtube.com/watch?feature=player_embedded&v=ugauQ769kVc#at=22). Це колекція, призначена для зберігання елементів в порядку, потрібному для їх обробки. На додаток до базових операцій інтерфейсу `Collection`, черга надає додаткові операції вставки, отримання та контролю.

3). Ознайомити з List та його різновидами. Пояснити методи List?

ArrayList - мабуть сама часто використовувана колекція. `ArrayList` інкапсулює в собі звичайний масив, довжина якого автоматично збільшується при додаванні нових елементів. Так як `ArrayList` використовує масив, то час доступу до елементу за індексом мінімально (На відміну від `LinkedList`). При видаленні довільного елемента зі списку, всі елементи знаходяться «правіше» зміщуються на одну клітинку вліво, при цьому реальний розмір масиву (його ємність, capacity) не змінюється. Якщо при додаванні елемента, виявляється, що масив повністю заповнений, буде створено новий масив розміром $(n * 3) / 2 + 1$, в нього будуть поміщені всі елементи зі старого масиву + новий, елемент.

LinkedList - двозвязаний список. Це структура даних, що складається з вузлів, кожен з яких містить як власне дані, так і два посилання («зв'язки») на наступний і попередній вузол списку.

Доступ до довільного елементу здійснюється за лінійний час (але доступ до першого і останнього елемента списку завжди здійснюється за константне час - посилання постійно зберігаються на перший і останній, так що додавання елемента в кінець списку зовсім не означає, що доведеться перебирати весь список у пошуках останнього елемента). В цілому ж, `LinkedList` в абсолютних величинах програє `ArrayList` і по споживаної пам'яті і по швидкості виконання операцій.

4). Призначення equals і hashCode?

HashCode

Якщо дуже просто, то **хеш-код** - це число. Насправді просто, чи не так? Якщо більш точно, то це бітовий рядок фіксованої довжини, отримана з масиву довільної довжини.

Приклад :

```
public class Main {
    public static void main(String[] args) {
        Object object = new Object();
        int hCode;
        hCode = object.hashCode();
        System.out.println(hCode);
    }
}
```

В результаті виконання програми в консоль виводиться ціле 10-ти значне число. Це число і є наш бітовий рядок фіксованої довжини. У java вона представлена у вигляді числа примітивного типу `int`, що дорівнює 4-м байтам, і може поміщати числа від -2147483648 до 2147483 647. На даному етапі важливо розуміти, що хеш-код це число, у якого є свої межі, який для java обмежений примітивним ціличисловим типом `int`.

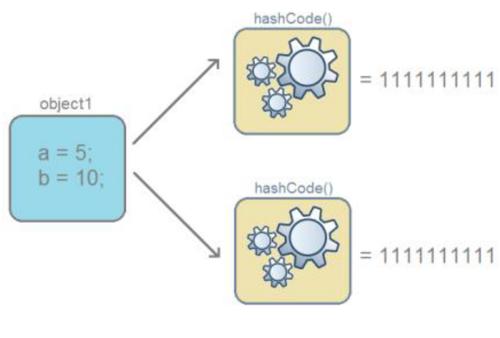
Тут головне зрозуміти, що:

- Якщо хеш-коди різні, то і вхідні об'єкти гарантовано різні.
- Якщо хеш-коди рівні, то вхідні об'єкти не завжди рівні.

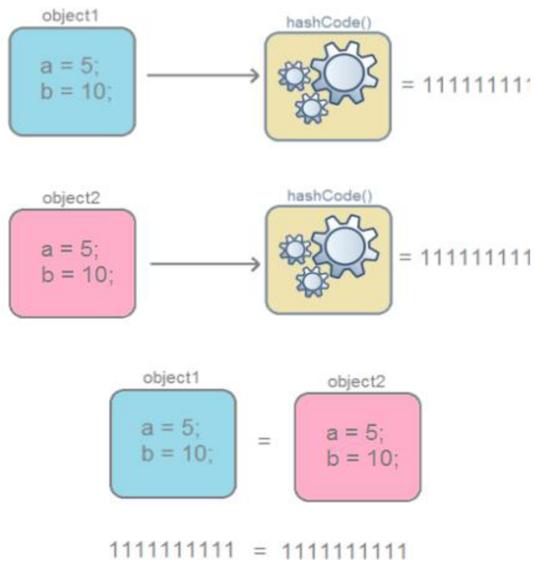
Ситуація, коли в різних об'єктів однакові хеш-коди називається - **колізією**.

Підіб'ємо підсумок! Спершу, щоб уникнути плутанини, визначимося з термінологією. Однакові об'єкти - це об'єкти одного класу з одинаковим вмістом полів.

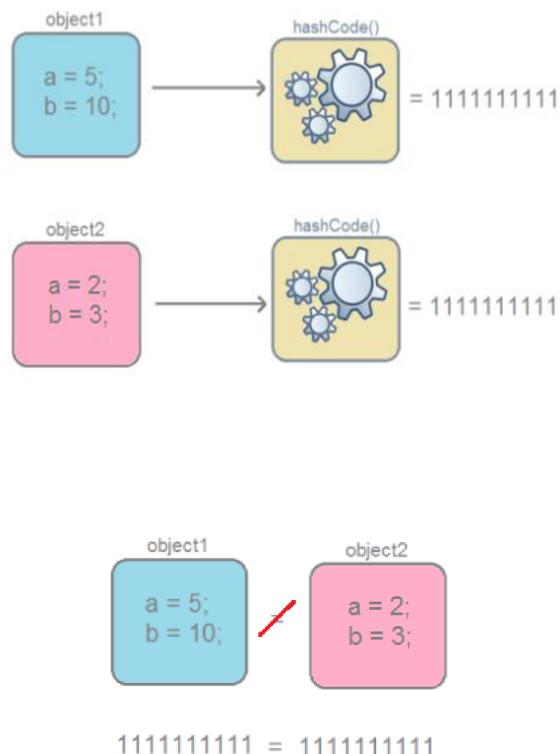
1. Для одного і того-ж об'єкта, хеш-код завжди буде одинаковим



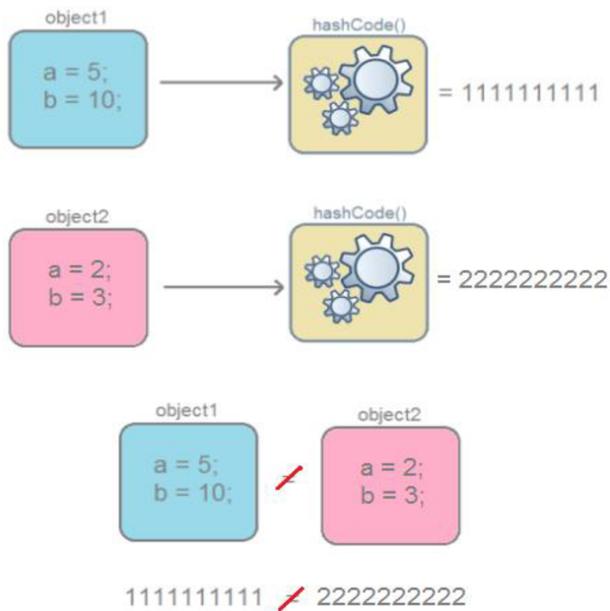
2. Якщо об'єкти однакові, то і хеш-коди однакові (але не навпаки)



3. Якщо хеш-коди рівні, то вхідні об'єкти не завжди рівні (колізія)



4. якщо хеш-коди різні, то і об'єкти гарантовано різні.



Equals

Почнемо з того, що в java, кожен виклик оператора new породжує новий об'єкт в пам'яті. Для ілюстрації створимо який-небудь клас, нехай він буде називатися "BlackBox".

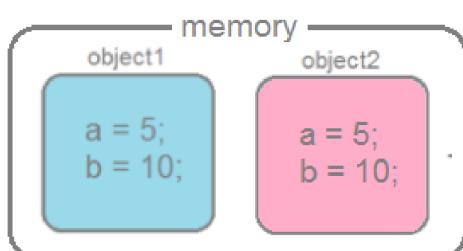
Виконуємо наступний код:

```
public class BlackBox {
    int varA;
    int varB;
    BlackBox(int varA, int varB){
        this.varA = varA;
        this.varB = varB;
    }
}
```

Створюємо клас для демонстрації BlackBox.

```
public class DemoBlackBox {
    public static void main(String[] args) {
        BlackBox object1 = new BlackBox(5, 10);
        BlackBox object2 = new BlackBox(5, 10);
    }
}
```

У другому прикладі, в пам'яті створюється два об'єкти.



Але, як ви вже звернули увагу, вміст цих об'єктів одинаковий, тобто еквівалентний. Для перевірки еквівалентності в класі Object існує метод equals (), який порівнює вміст об'єктів і виводить значення типу boolean -true, якщо вміст еквівалентний, і false - якщо ні.

```
object1.equals (object2); // повинно бути true, оскільки вміст об'єктів еквівалентний
```

! equals і hashCode тісно пов'язані між собою, оскільки хеш-код обчислюється на підставі вмісту об'єкта (значення полів) і якщо у двох об'єктів одного і того ж класу вміст одинаковий, то і хеш-коди повинні бути однакові. Іншими словами:

```
object.equals (object) // повинно бути true
object1.hashCode () == object2.hashCode () // повинно бути true
```

Я написав "повинно бути", бо якщо ви виконаете попередній приклад, то насправді результатом виконання всіх операцій буде false. Для пояснення причин, заглянемо в вихідні коди класу Object.

клас Object

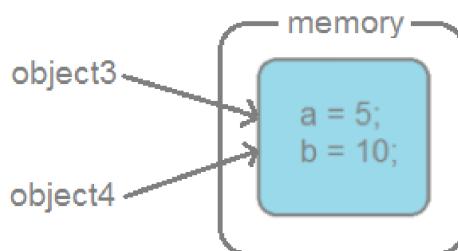
Як відомо, всі java-класи успадковуються від класу Object. У цьому класі вже визначені методи hashCode () і equals () .

Створюючи свій клас, ви автоматично успадковуєте всі методи класу Object. І в ситуації, коли у вашому класі не перевизначені (overriding) hashCode () і equals (), то використовується їх реалізація з Object. Розглянемо вихідний код методу equals () в класі Object:

```
public boolean equals (Object obj) {
    return (this == obj);
}
```

При порівняння об'єктів, операція "==" поверне true лише в одному випадку - коли посилання вказують на один і той-же об'єкт. В даному випадку не враховується вміст полів. Виконавши наведений нижче код, equals поверне true.

```
public class Demo BlackBox {
    public static void main (String [] args) {
        BlackBox object3 = new BlackBox (5, 10);
        BlackBox object4 = object3; // змінна object4 посилається на
                                // той-же об'єкт що і змінна object3
        object3.equals (object4) // true}
    }
}
```



Тепер зрозуміло, чому Object.equals () працює не так як потрібно, адже він порівнює посилання, а не вміст об'єктів. Далі на черзі hashCode (), який теж працює не так як належить. Заглянемо у вихідний код методу hashCode () в класі Object:

```
public native int hashCode();
```

При обчисленні хеш-коду для об'єктів класу Object за замовчуванням використовується Park-Miller RNG алгоритм. В основу роботи даного алгоритму покладено генератор випадкових чисел. Це означає, що при кожному запуску програми в об'єкта буде різний хеш-код.

Виходить, що використовуючи реалізацію методу hashCode () від класу Object, ми при кожному створенні об'єкта класу new BlackBox (), будемо отримувати різні хеш-коди. Мало того, перезапуску програму, ми будемо отримувати абсолютно різні значення, оскільки це просто випадкове число.

Але, як ми пам'ятаємо, повинно виконуватися правило: "якщо у двох об'єктів одного і того ж класу вміст однакове, то і хеш-коди повинні бути однакові". Тому, при створенні користувальницького класу, прийнято перевизначати методи hashCode () і equals () таким чином, що б враховувалися поля об'єкта.

Це можна зробити вручну або скориставшись засобами генерації вихідного коду в IDE. Наприклад, в Eclipse це Source → Generate hashCode () and equals () ... У підсумку, клас BlackBox набуває вигляду:

```
public class BlackBox {
    int varA;
    int varB;
    BlackBox(int varA, int varB){
        this.varA = varA;
        this.varB = varB;
    }
    @Override public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + varA;
        result = prime * result + varB;
        return result;
    }
    @Override public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null) return false;
        if (getClass() != obj.getClass()) return false;
        BlackBox other = (BlackBox) obj;
        if (varA != other.varA) return false;
        if (varB != other.varB) return false;
        return true;
    }
}
```

Тепер методи hashCode () і equals () працюють коректно і враховують вміст полів об'єкта:

```
object1.equals(object2); //true object1.hashCode() == object2.hashCode(); //true
```

Домашня робота:

Використовуючи Домашнє завдання до уроку 6 (де ми створювали машину), в методі мейн створюємо список List(Яку саме реалізацію даного інтерфейсу , вибирайте самі). Тестуємо всі методи List над даним списком. Результати виводимо на екран.

Заняття 9

1).Різниця між ArrayList vs LinkedList?

LinkedList забезпечує кращу продуктивність в наступних випадках:

1. При операціях додавання / видалення елементів на початку списку за індексом.

2. При операціях додавання / видалення елементів усередині списку по ітератори (за умови, що цей ітератор був якимось чином отриманий заздалегідь).
3. При доступі до першого / останнього елемента списку (getFirst () / getLast () / removeFirst () / removeLast () і т.д.)

ArrayList - мабуть сама часто використовувана колекція. ArrayList інкапсулює в собі звичайний масив, довжина якого автоматично збільшується при додаванні нових елементів. Так як ArrayList використовує масив, то час доступу до елементу за індексом мінімально (На відміну від LinkedList). При видаленні довільного елемента зі списку, всі елементи знаходяться «правіше» зміщуються на одну клітинку вліво, при цьому реальний розмір масиву (його ємність, capacity) не змінюється. Якщо при додаванні елемента, виявляється, що масив повністю заповнений, буде створено новий масив розміром $(n * 3) / 2 + 1$, в нього будуть поміщені всі елементи зі старого масиву + новий, елемент.

LinkedList - двусвязного список. Це структура даних, що складається з вузлів, кожен з яких містить як власне дані, так і два посилання («зв'язки») на наступний і попередній вузол списку. Доступ до довільного елементу здійснюється за лінійний час (але доступ до першого і останнього елемента списку завжди здійснюється за константне час - посилання постійно зберігаються на перший і останній, так що додавання елемента в кінець списку зовсім не означає, що доведеться перебирати весь список у пошуках останнього елемента). В цілому ж, LinkedList в абсолютних величинах програє ArrayList і по споживаної пам'яті і по швидкості виконання операцій.

2). Ітератори, призначення та методи?

Ітератор - це об'єкт, який дозволяє програмісту пробігти по елементах колекції. «Ну і що?» - Скажуть деякі - «Я можу це зробити за допомогою звичайного циклу!».

Так, дійсно, більшість стандартних колекцій з пакету java.util надають можливість вибірки елемента за його індексом.

Як обійти елементи Set'a? Ось тут то нам і приде на допомогу ітератор. Всі колекції з java.util реалізують інтерфейс Collection, який, у свою чергу, розширяє інтерфейс Iterable. Ось воно, наше рішення! В інтерфейсі Iterable описаний тільки один метод: **Iterator iterator ()**

Приклад :

```
public class Main {
    public static void main(String[] args) {
        ArrayList arrayList = new ArrayList();
        arrayList.add("element 1");
        arrayList.add("element 2");
        arrayList.add("element 3");
        arrayList.add("element 4");
        arrayList.add("element 5");

        ListIterator it = arrayList.listIterator();

        System.out.println("Forward iteration :");
        while (it.hasNext()){
            System.out.println(it.next());
        }
    }
}
```

3). Закріплення навиків роботи з введенням даних з клавіатури.

Для введення даних використовується клас Scanner з бібліотеки пакетів Java.

Цей клас треба імпортувати в тій програмі, де він буде використовуватися. Це робиться до початку відкритого класу в коді програми.

У класі є методи для читання чергового символу заданого типу зі стандартного потоку введення, а також для перевірки існування такого символу.

Для роботи з потоком введення необхідно створити об'єкт класу Scanner, при створенні вказавши, з яким потоком введення він буде пов'язаний. Стандартний потік вводу (клавіатура) в Java

представлений об'єктом - `System.in`. А стандартний потік виводу (дисплей) - уже знайомим вам об'єктом `System.out`.

Метод `hasNextDouble()`, застосований об'єкту класу `Scanner`, перевіряє, чи можна вважати з потоку введення дійсне число типу `double`, а метод `nextDouble()` - читає його. Якщо спробувати вважати значення без попередньої перевірки, то під час виконання програми можна отримати помилку (відладчик заздалегідь таку помилку не виявить). Наприклад, спробуйте в представленої далі програмі ввести якесь дійсне число:

Є також метод `nextLine()`, що дозволяє читувати цілу послідовність символів, тобто рядок, а, значить, отримане через цей метод значення потрібно зберігати в об'єкті класу `String`. У наступному прикладі створюється два таких об'єкти, потім в них по черзі записується введення користувача, а далі на екран виводиться один рядок, отримана об'єднанням введених послідовностей символів.

Домашня робота:

Моделюємо верховну раду. Створити клас Людина, описати його наступними полями : вага, ріст, додати джентельменський набір. Створити клас депутат , унаслідувати його від Людини. Описати його такими полями: прізвище, ім'я, вік, хабарник(Буліановське), розмір хабаря(не передавати в конструктор). Додати джентельменський набір . Додати метод : дати хабар(), в якому передбачити наступне :

- якщо поле хабарник `false` - то вивести на консоль :" Цей депутат не бере хабарів", якщо умова не виконується ,

то ввести з консолі суму хабаря яку ви даете,якщо це сума більша 5000, вивести на консоль " Миліція увянить депутата",якщо не більша то занести в поле класу хабар дану суму.

Створити клас фракція ,якому описати наступні методи :

- додати депутата(вводимо з консолі)
- видалити депутата(вводимо з консолі)
- вивести всіх хабарників у фракції
- вивести найбільшого хабарника у фракції
- вивести всіх депутатів фракції
- очистити всю фракцію від депутатів

Створити клас верховна рада і реалізувати в ньому наступні методи(дозволено створити тільки один екземпляр даного класу(singleton)):

- додати фракцію
- видалити фракцію
- вивести всі фракції
- вивести конкретну фракцію
- додати депутата до конкретної фракції
- видалити депутата(вводимо з консолі)
- вивести всіх хабарників у фракції
- вивести найбільшого хабарника у фракції
- вивести всіх депутатів фракції

Створити клас Мейн в якому описати наступне консольне меню:

- Введіть 1 щоб додати фракцію

- Введіть 2 щоб видалити фракцію
- Введіть 3 щоб очистити фракцію
- Введіть 4 щоб вивести фракції
- Введіть 5 щоб вивести фракцію
- Введіть 6 щоб додати депутата в фракцію
- Введіть 7 щоб видалити депутата з фракції
- Введіть 8 щоб вивести список хабарників
- Введіть 9 щоб вивести найбільшого хабарника

Заняття 10 – Заняття 11

1. Розказати про Set та його різновиди HashSet, TreeSet, Пояснити різницю між ними. ?

HashSet - колекція, що не дозволяє зберігати однакові об'єкти (як і будь Set). HashSet інкапсулює в собі об'єкт HashMap (тобто використовує для зберігання хеш-таблицю).

Як більшість читачів, ймовірно, знають, хеш-таблиця зберігає інформацію, використовуючи так званий механізм хеширования, в якому вміст ключа використовується для визначення унікального значення, званого хеш-кодом. Цей хеш-код потім застосовується в якості індексу, з яким асоціюються дані, доступні з цього ключу. Перетворення ключа в хеш-код виконується автоматично - ви ніколи не побачите самого хеш-коду. Також ваш код не може напряму індексувати хеш-таблицю. Вигода від хеширования полягає в тому, що воно забезпечує константне час виконання методів add (), contains (), remove () і size (), навіть для великих наборів.

Якщо Ви хочете використовувати HashSet для зберігання об'єктів СВОЇХ класів, то ви ПОВИННІ перевизначити методи hashCode () і equals (), інакше два логічно-однакових об'єкта будуть вважатися різними, так як при додаванні елемента в колекцію буде викликатися метод hashCode () класу Object (який швидше-всього поверне різний хеш-код для ваших об'єктів).

Важливо відзначити, що клас HashSet не гарантує впорядкованості елементів, оскільки процес хеширования сам по собі звичайно не породжує сортованих наборів. Якщо вам потрібні сортовані набори, то кращим вибором може бути інший тип колекцій, такий як клас TreeSet.

LinkedHashSet - підтримує зв'язний список елементів набору в тому порядку, в якому вони вставлялися. Це дозволяє організувати впорядковану ітерацію вставки в набір. Тобто, коли йде перебір об'єкта класу LinkedHashSet із застосуванням ітератора, елементи витягаються в тому порядку, в якому вони були додані.

TreeSet - колекція, яка зберігає свої елементи у вигляді упорядкованого за значеннями дерева. TreeSet інкапсулює в собі TreeMap, який в свою чергу використовує збалансоване бінарне червоно-чорне дерево для зберігання елементів. TreeSet хороший тим, що для операцій add, remove і contains потрібно гарантований час $\log(n)$.

PriorityQueue - єдина пряма реалізація інтерфейсу Queue (не рахуючи LinkedList, який більше є списком, ніж чергою).

Ця черга впорядковує елементи або за їх натуральному порядку (використовуючи інтерфейс Comparable), або за допомогою інтерфейсу Comparator, отриманому в конструкторі.

2. Пояснити призначення Comparable та Comparator?

Інтерфейс **Comparable** визначає логіку порівняння об'єкта певного посилального типу всередині своєї реалізації і якщо немає доступу до ісходникам її неможливо змінити. Інтерфейс **Comparator** позволяє визначити логіку порівняння об'єктів певного посилального типу поза реалізації цього типу і цю логіку можна в будь-який момент підмінити.

Приклад :

Comparable

```
package Lesson10_Set_Comparator_Comparable;

import java.io.ObjectInputStream.GetField;

class Person implements Comparable<Person>{
    private String firstName;
    private String lastName;
    private int age;
    static int count;

    public Person(String fname, String lname, int age) {
        this.age = age;
        this.firstName = fname;
        this.lastName = lname;
    }

    @Override
    public int compareTo(Person o) {

        int l=this.getLastName().compareTo(o.getLastName());
        int f=this.getFirstName().compareTo(o.firstName);
        if(l==0 && f==0 && this.getAge()==o.getAge()){
            return 0;
        }
        else if(this.getAge()==o.getAge()){
            return 0;
        }
        else if(this.getAge()> o.getAge()){
            return 1;
        };
        return -1;
    }
}
```

Comparator:

```
public class AgeComparator implements Comparator<Person> {

    @Override
    public int compare(Person o1, Person o2) {
        if (o1.getAge() > o2.getAge())
            return 1;
        else if (o1.getAge() < o2.getAge())
            return -1;
        else
            return 0;
    }
}
```

```

        return -1;
    else
        return 0;
}
}

```

Домашня робота:

- 1) Створити клас(довільно який, наприклад Person), описати в ньому мінімум два поля, одне з яких String, інше числове(довільно яке). Створити в мейн методі Set, List-реалізацію на ваш розсуд. Наповнити їх обєктами Person. Використати Comparator і Comparable для сортування за цими полями. Вивести спершу невідсортований список, потім відсортований на консоль.
- 2) Реалізовуємо консольну програму. Створити клас Commodity. Описати даний клас: поля методи. Повинні бути такі методи:

- Додати товар
- Видалити товар
- Замінити товар
- Сортувати за назвою
- Сортувати за довжиною
- Сортувати за ширину
- Сортувати за вагою
- Виводимо і-тий елемент колекції(який ми вводимо з консолі(використовуємо Scanner))
- Вийти з програми(підказка System.exit)

Для меню використати Switch. Продемонструвати як виконується кожен метод над списком і виводити список після змін в ньому.

Заняття 12

1. Розказати про Мар та його різновиди HashMap, TreeMap, LinkedHashMap?

Інтерфейс Map співвідносить унікальні ключі зі значеннями. Ключ - це об'єкт, який ви використовуєте для подальшого вилучення даних. Задаючи ключ і значення, ви можете поміщати значення в об'єкт карти. Після того як це значення збережено, ви можете отримати його по ключу. Інтерфейс Map - це узагальнений інтерфейс, оголошений так, як показано нижче.

interface Map<K, V> Тут K вказує тип ключів, а V - тип збережених значень.

Ієрархія класів дуже схожа на ієрархію Set'a:

HashMap - заснований на хеш-таблицях, реалізує інтерфейс Map (що передбачає зберігання даних у вигляді пар ключ / значення). Ключі і значення можуть бути будь-яких типів, у тому числі і null. Дані реалізація не дає гарантій щодо порядку елементів з плином часу. Хороша стаття <http://habrahabr.ru/post/128017/>

LinkedHashMap - розширяє клас HashMap. Він створює зв'язний список елементів у карті, розташованих в тому порядку, в якому вони вставлялися. Це дозволяє організувати перебір карти в порядку вставки. Тобто, коли відбувається ітерація по колекційному поданням об'єкта класу LinkedHashMap, елементи будуть повертатися в тому порядку, в якому вони вставлялися. Ви також можете створити об'єкт класу LinkedHashMap, який повертає свої елементи в тому порядку, в якому до них востаннє здійснювався доступ.

Рекомендую так само прочитати <http://habrahabr.ru/post/129037/>

TreeMap - розширює клас AbstractMap і реалізує інтерфейс NavigableMap. Він створює колекцію, яка для зберігання елементів застосовує дерево. Об'єкти зберігаються в відсортованому порядку за зростанням. Час доступу та вилучення елементів досить мало, що робить клас TreeMap близьким вибором для зберігання великих обсягів відсортованої інформації, яка повинна бути швидко знайдена.

Моя стаття про TreeMap <http://www.quizful.net/post/Java-TreeMap>

WeakHashMap - колекція, яка використовує слабкі посилання для ключів (а не значень). Слабка посилання (англ. Weak reference) - специфічний вид посилань на динамічно створювані об'єкти в системах із збіркою сміття. Відрізняється від звичайних посилань тим, що не враховується складальником сміття при виявленні об'єктів, що підлягають видаленню. Посилання, які не є слабкими, також іноді називають «сильними».

Домашня робота :

Створити клас Зооклуб , як поля прописати в ньому Map. Створити клас Person , який описати двома полями : ім'я , вік. Створити клас Animal , який описати двома полями : тип тваринки(кіт, пес), ім'я тваринки. В класі Зооклуб як поле прописати наступне:

Map<Person, List<Pet>> map;

Провести ініціалізацію даних полів в конструкторі

```
map = new HashMap<>();
```

Реалізувати консольне меню, таким чином щоб можна було:

- Додати учасника клубу
- Додати тваринку до учасника клубу
- Видалити тваринку з учасника клубу
- Видалити учасника клубу
- Видалити конкретну тваринку зі всіх власників
- Вивести на екран зооклуб
- Вийти з програми

Використати для побудови меню Switch.

Заняття 13 (Тест по Колекціях)

Доманя робота : Доробити завдання , що не було закінчено на занятті.

Заняття 14

1. Ознайомлення з ієрархією Throwable.

Тут основними класами є **Throwable**, **Error**, **Exception** і **RuntimeException**. Саме з ними пов'язана вся "магія".

Справа в тому, що в java є два типи винятків: **checked i unchecked**.

1. **Checked** виключення, це ті, які повинні оброблятися блоком catch або описуватися в сигнатурі методу. Unchecked можуть не оброблятися і не бути описаними.
2. **Unchecked** виключення в Java - успадковані від RuntimeException, checked - від Exception (не включаючи unchecked).

Тобто якщо у вашому коді є ділянка який може кинути **checked** виняток то ви зобов'язані або укласти його в конструкцію **try / catch** або оголосити в сигнатурі методу **"throws SomeException"** але в даному випадку обробка виключення делегується на рівень вище. У будь-якому випадку його потрібно буде перехопити. В іншому випадку програма просто не скомпілюється.

Так чому не всі винятки перевіряються? Справа в тому, що якщо перевіряти кожне місце, де теоретично може бути помилка, то ваш код сильно розростеться, стане погано читаним. Наприклад в будь-якому місці, де відбувається ділення чисел, потрібно було б перевіряти на ArithmeticException, тому що можливе ділення на нуль. Цю опцію (відловлювати / не перевіряти виключення) творці мови залишили програмісту на його розсуд.

Повернемося до класів винятків:

1. RuntimeException

IndexOutOfBoundsException - викидається, коли індекс деякого елемента в структурі даних (масив / колекція) не потрапляє в діапазон наявних індексів.

NullPointerException - посилення на об'єкт, до якого ви звертаєтесь зберігає null /

ClassCastException - Помилка приведення типів. Всякий раз при приведенні типів робиться перевірка на можливість приведення (перевірка здійснюється за допомогою instanceof).

ArithmaticException - кидається коли виконуються неприпустимі арифметичні операції, наприклад розподіл на нуль.

2. Error

ThreadDeath - викликається при несподіваній зупинці потоку за допомогою методу Thread.stop () .

StackOverflowError - помилка переповнення стека. Часто виникає в рекурсивних функціях через неправильне умови виходу.

OutOfMemoryError - помилка переповнення пам'яті.

2. Пояснити поняття Exception and Error?

Виняток - це проблема (помилка) виникає під час виконання програми. Винятки можуть виникати у багатьох випадках, наприклад:

- Користувач ввів некоректні дані.
- Файл, до якого звертається програма, не знайдений.
- Мережеве з'єднання з сервером було загублено під час передачі даних.

Обробка виняткових ситуацій (exception handling) - механізм мов програмування, призначений для опису реакції програми на помилки часу виконання та інші можливі проблеми (виключення), які можуть виникнути при виконанні програми і призводять до неможливості (безглуздості) подальшого відпрацювання програмою її базового алгоритму.

Error- це підклас, який показує серйозні проблеми які виникають під час виконання програми. Більшість з цих помилок сигналізують про ненормальний хід виконання програми, тобто про якісь

критичні проблеми. Ці помилки не рекомендується відзначати в методах допомогою throws-оголошення, тому вони також дуже часто називаються unchecked.

3. Ознайомлення з try, catch, throw, throws. ?

У Java є п'ять ключових слів для роботи з винятками:

try - це ключове слово використовується для позначки початку блоку коду, який потенційно може привести до помилки.

catch - ключове слово для відмітки початку блоку коду, призначеного для перехоплення і обробки виключень.

finally - ключове слово для відмітки початку блоку коду, яке є додатковим. Цей блок поміщається після останнього блоку 'catch'. Управління зазвичай передається в блок 'finally' в будь-якому випадку.

throw - служить для генерації винятків.

throws - ключове слово, яке прописується в сигнатурі методу, і позначає що метод потенційно може викинути виняток з вказаним типом.

4. Як створити свій Exception?

Створюємо клас, extends Exception.

```
public class MyException extends Exception{
    public MyException (String msg) {
        super (msg);
    }
}
```

Домашня Робота :

Створити свій Exception(MyException). Створити клас Methods, Описати в ньому методи , які б розраховували додавання, віднімання, множення, ділення двох змінних . При цьому врахувати :

- Якщо $a < 0, b < 0$ викидаємо IllegalArgumentException
- Якщо $a = 0, b \neq 0$ викидаємо ArithmaticException
- Якщо $a \neq 0, b = 0$ викидаємо ArithmaticException
- Якщо $a = 0, b = 0$ викидаємо IllegalAccessException
- Якщо $a > 0, b > 0$ викидаємо MyException

Протестувати всі можливі варіанти, і вивести все на консоль.

Заняття 15

1. Запис та читання з файлів.?

ІО API — (Input & Output) - Дуже часто доводиться отримувати якийсь потік даних, а потім якось їх обробляти і відправляти далі. Наприклад, користувач ввів логін і пароль, програма в свою чергу повинна отримати ці дані, обробити і зберегти файл. Для цих цілей можна використовувати ІО.

У Java бібліотека IO API знаходиться в пакеті `java.io` і для того щоб почати використовувати IO достатньо імпортувати дану бібліотеку в ваш клас.

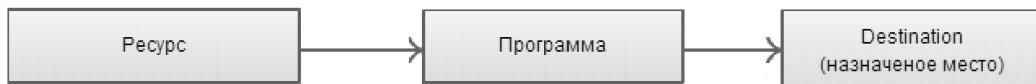
Input and Output - Призначення

У `java.io` існують так звані потоки введення та виведення (`InputStream` and `OutputStream`).

В основному `java.io` призначений для читання і запису даних в ресурс:

- 1) файл;
- 2) при роботі з мережевим підключенням;
- 3) `System.err`, `System.in`, `System.out`;
- 4) при роботі з буфером.

Процес читання даних з ресурсу і запис їх у призначене місце показаний на малюнку нижче.



Програма яка повинна зчитати дані з потоку і записати в потік показана на малюнку нижче.



Java IO - Здібності

У Java IO багато класів які в основному працюють з потоками читання і запису, і вирішують різні завдання:

- Отримання доступу до файлів;
- Отримання мережевого з'єднання;
- Робота з буфером;
- Доступ до внутрішньої буферу пам'яті;
- Міжпоточного спілкування;
- Парсинг даних;
- Читання і запис тексту;
- Читання і запис примітивних даних (`long`, `int`, `float` ...);
- Читання і запис об'єктів.

Всі ці можливості вам надасть Java IO.

Класи Java IO API

Базові

- `InputStream` / `OutputStream`
- `Reader` / `Writer`
- `InputStreamReader` / `OutputStreamWriter`

Масиви

- `ByteArrayInputStream` / `ByteArrayOutputStream`
- `CharArrayReader` / `CharArrayWriter`

Files

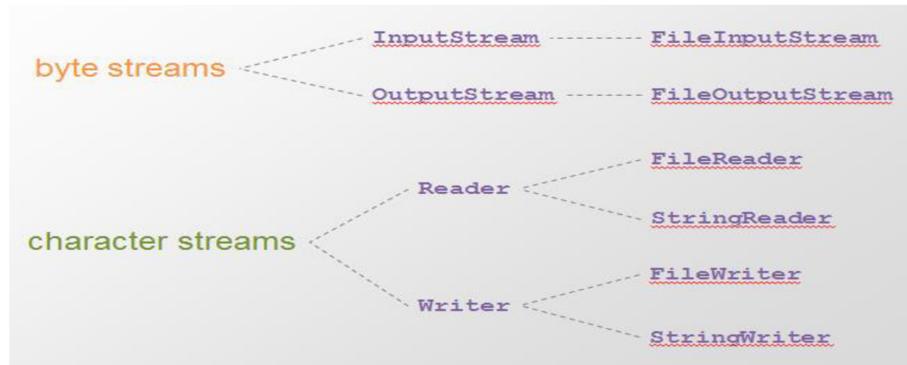
- `FileInputStream` / `FileOutputStream`
- `RandomAccessFile` / `RandomAccessFile`
- `FileReader` / `FileWriter`

Буферизація

- `BufferedInputStream` / `BufferedOutputStream`

- BufferedReader / BufferedWriter

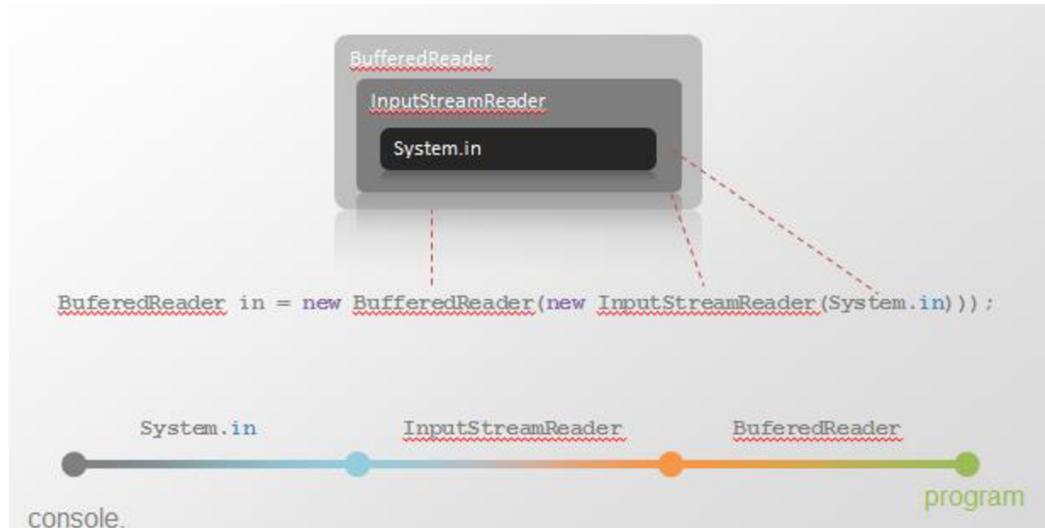
Байтові та Символьні потоки в Java



Класи бібліотеки ІО а Java

	Byte Based Input	Output	Character Based Input	Output
Basic	InputStream	OutputStream	Reader InputStreamReader	Writer OutputStreamWriter
Arrays	ByteArrayInputStream	ByteArrayOutputStream	CharArrayReader	CharArrayWriter
Files	FileInputStream RandomAccessFile	FileOutputStream RandomAccessFile	FileReader	FileWriter
Pipes	PipedInputStream	PipedOutputStream	PipedReader	PipedWriter
Buffering	BufferedInputStream	BufferedOutputStream	BufferedReader	BufferedWriter
Filtering	FilterInputStream	FilterOutputStream	FilterReader	FilterWriter
Parsing	PushbackInputStream StreamTokenizer		PushbackReader LineNumberReader	
Strings			StringReader	StringWriter
Data	DataInputStream	DataOutputStream		
Data - Formatted		PrintStream		PrintWriter
Objects	ObjectInputStream	ObjectOutputStream		
Utilities	SequenceInputStream			

Послідовність взаємодії потоків



2. Робота зі стрічками?

Робота зі стрічками зводиться до того щоб вміти їх розділити. Для цього можна використовувати StringTokenizer, Split.

```
 StringTokenizer st =new StringTokenizer("string tokenizer example");
System.out.println("tokens count: " + st.countTokens());
// iterate through st object to get more tokens from it
while (st.hasMoreElements()) {
String token = st.nextElement().toString();
System.out.println("token = " + token);
}
```

Фрагмент кода для String[] split(String regEx)

```
String str = "st1-st2-st3";
String delimiter = "-";
String[] temp;
temp = str.split(delimiter);
for(int i=0; i < temp.length ; i++)
System.out.println(temp[i]);
```

Получается:

stl

st2

st3

Фрагмент кода для String[] split(String regEx, int limit)

```
String str = "st1-st2-st3";
String delimiter = "-";
String[] temp;
temp = str.split(delimiter, 2);
for(int i = 0; i < temp.length ; i++)
System.out.println(temp[i]);
```

Получается:

st1

st2-st3

Домашня робота:

Використовуючи наш Зооклуб(Заняття 12), який ми робили на попередньому занятті додати ще методи

- Дописати в блокнот
- Перезаписати в блокнот
- Зчитати з блокноту на консоль

Заняття 16

1. Що таке Сереалізація?

Серіалізація це процес збереження стану об'єкта в послідовність байт; десеріалізацію це процес відновлення об'єкта, з цих байт.

Серіалізація - це перетворення екземпляра класу у форму, придатну для його збереження (наприклад у файл, в БД або дя передачі по мережі). Серіалізовани об'єкти можна потім відновити (десеріалізувати). Так от, усі поля класу, які позначені як transient НЕ будуть зберігатися.

Зазвичай в таких полях зберігається проміжний стан об'єкта, яке, наприклад, простіше вирахувати, ніж серіалізувати, а потім десеріалізувати. Інший приклад такого поля - посилання на екземпляр об'єкта, який не вимагає серіалізації або не може бути серіалізовани.

2. Модифікатор transient?

Якщо ми не хочемо щоб поле считувалось при серіалізації, тоді помічаємо його модифікатором transient.

Домашня робота :

Створити клас Employee, описати даний клас наступними полями: name, id, salary. Дані поля повинні бути private. Описати getters and setters. Створити клас Methods де описати всього два методи serialize() and deserialize(). В даних методах повинно бути бути використано FileInputStream/FileOutputStream , ObjectInputStream/ObjectOutputStream.

- 1).Створити метод Main створити екземпляр класу Employee і провести серіалізацію та десеріалізацію. Результат Десеріалізації вивести на консоль.
- 2). Помітити поле salary модифікатором transient і провести серіалізацію і десеріалізацію. Результат вивести на екран.

Заняття 17

1. Ознайомлення з внутрішніми класами, пояснити зв'язки зовнішній-внутрішній і навпаки.?

Вкладений клас - це клас, який оголошений всередині оголошення іншого класу.

```
class OuterClass {
    ...
    static class StaticNestedClass {
        ...
    }
    ...
}
```

Внутрішні класи в Java діляться на такі три види:

- внутрішні класи-члени (member inner classes);
- локальні класи (local classes);
- анонімні класи (anonymous classes).

Внутрішні класи-члени асоціюються не з самим зовнішнім класом, а з його екземпляром. При цьому вони мають доступ до всіх його полів і методів.

```
public class Users {
    ...
    public class Query {
        private Query() { ... }
        public void setLogin(String login) { ... }
        public void setCreationDate(Date date) { ... }
        public List<User> list() { ... }
        public User single() { ... }
    }

    public Query createQuery() { return new Query(); }
}
```

Локальні класи (local classes) - визначаються в блоці Java коду. На практиці найчастіше оголошення відбувається в методі деякого іншого класу. Хоча оголошувати локальний клас можна всередині статичних і нестатичних блоків ініціалізації.

```
public class Handler {
    public void handle(String requestPath) {
        class Path {
            List<String> parts = new ArrayList<String>();
            String path = "/";
            Path(String path) {
                if (path == null) return;
                this.path = path;
                for (String s : path.split("/"))
                    if (s.trim().length() > 0) this.parts.add(s);
            }
            int size() { return parts.size(); }
            String get(int i) { return i > this.parts.size() - 1 ? null : this.parts.get(i); }
            boolean startsWith(String s) { return path.startsWith(s); }
        }

        Path path = new Path(requestPath);

        if (path.startsWith("/page")) {
            String pageId = path.get(1);
            ...
        }
        if (path.startsWith("/post")) {
            String categoryId = path.get(1);
            String postId = path.get(2);
        }
        ...
    }
}
```

```

    }
}
```

У локальних класів є безліч обмежень:

- вони видні тільки в межах блоку, в якому оголошенні;
- вони не можуть бути оголошенні як private, public, protected або static;
- вони не можуть мати всередині себе статичних оголошень (полів, методів, класів);
винятком є константи (static final);

Анонімні класи - є важливою підмогою в повсякденному житті Java-програмістів. Анонімний клас (anonymous class) - це локальний клас без імені.

```

new Thread(new Runnable() {
    public void run() {
        ...
    }
}).start();
```

2. Пояснити призначення анонімних та локальних класів. ?

Використання анонімних класів виправдано в багатьох випадках, зокрема коли:

- тіло класу є дуже коротким;
- потрібен тільки один екземпляр класу;
- клас використовується в місці його створення або відразу після нього;
- ім'я класу не важливо і не полегшує розуміння коду.

Створити екземпляр статичного класу :

```
OuterClass.StaticNestedClass instance = new OuterClass.StaticNestedClass();
```

Створити екземпляр внутрішнього класу:

```
Outer outer = new Outer();
Outer.Inner inner = outer.new Inner();
```

Домашня робота :

Створити інтерфейс Iterator, в якому оголосити метод hasNext(), next(). Створити клас Collection, в якого оголосити як поле масив типу Object. Створити конструктор з визначеними параметрами куди передати даний масив. Створити два внутрішніх класи, які імплементуватимуть інтерфейс Iterator. Перевизначити методи так щоб:

- Перший клас виводив значення масиву від першого до останнього. Зробити заміну непарних елементів на нуль і вивести на екран даний масив.
- Другий клас виводив від останнього до першого значення через одну цифру.

Створити анонімний клас який повинен :

- Прогнати від останнього до першого елементу масиву. Перевірити кожен третій елемент масиву від останнього чи він непарний, якщо так то вивести дані елементи на консоль.

Створити Локальний клас який повинен:

- Пройтись від першого до останнього елементу масиву, перевірити кожен п'ятий елемент масиву, якщо він парний то відняти від нього число 100, і вивести на консоль ці числа.

Створити статичний клас який повинен:

- Пройтись від першого до останнього елемента масиву. Перевірити кожен другий елемент масив на парність, якщо він парний, тоді зробити з нього непарний і вивести дані елементи на консоль.

Заняття 18

1. Ознайомити з поняттям потоку, його станами?

Якщо сказати просто, то потік (thread) - це шлях програмного виконання(це те що йде в ході виконання програми). Більшість програм, написаних сьогодні, запускаються одним потоком, проблеми починають виникати, коли кілька подій або дій повинні відбутися в один час. Припустимо, наприклад, програма не здатна малювати картинку поки виконує читання натиснення клавіш. Програма повинна приділяти всю свою увагу клавіатурі, внаслідок чого відсутня можливість обробляти більше одного події одночасно. Ідеальним рішенням для цієї проблеми може служити можливість виконання двох або більше розділів програми в один час. Потоки дозволяють нам це зробити.

Багатопотокові програми надають міць при запуску багатьох потоків в рамках однієї програми. З логічної точки зору, многопоточність означає, що кілька рядків з однієї і тієї ж програми можуть бути виконані в один і той же час, однак, це не те ж саме, що запустити програму двічі і сказати, що кілька рядків коду виконуються в один час .

Потоки мають такі стани:

- **Новий** (це коли тільки створили екземпляр класу Thread)
- **Живий** або працездатний (перходить в цей стан після запуску методу start (), але це не означає що потік вже працює! Або ж він може перейти в цей стан з стану працюючий або блокований)
- **Працюючий** (це коли метод run () почав виконуватися)
- **Очікуючий** (waiting) / **Заблокований** (blocked) / **Сплячий** (sleeping). Ці стани характеризують потік що не готовий до роботи. Я об'єднав ці стани тому всі вони мають спільну рису - потік ще живий (alive), але в даний час не може бути виконаний.

Іншими словами потік вже не працює, але він може повернутися в робочий стан. Потік може бути заблокований, це може означати що він чекає звільнення якихось ресурсів. Потік може спати, якщо зустрівся метод sleep (long s), або ж він може очікувати, якщо зустрівся метод wait (), він буде чекати поки не викликом методу notify () або notifyall () .

- **Мертвий** (стан коли метод run () завершив свою роботу)

2. Розказати про клас Thread, інтерфейс Runnable ?

Коли запускається будь-який додаток, то починає виконуватися потік, званий головним потоком (**main**). Від нього породжуються дочірні потоки. Головний потік, як правило, є останнім потоком, завершальним виконання програми.

Незважаючи на те, що головний потік створюється автоматично, ім можна управляти через об'єкт класу **Thread**. Для цього потрібно викликати метод **currentThread ()**, після чого можна управляти потоком.

Клас **Thread** містить кілька методів для управління потоками.

- getName ()** - отримати ім'я потоку
- getPriority ()** - отримати пріоритет потоку
- isAlive ()** - визначити, чи виконується потік
- join ()** - очікувати завершення потоку
- run ()** - запуск потоку
- sleep ()** - призупинити потік на заданий час
- start ()** - запустити потік викликом методу start ()

Отримаємо інформацію про головному потоці і помінямо його ім'я.

```
Thread mainthread = Thread.currentThread();
textViewInfo.setText("Теперішній потік: " + mainthread.getName());
```

// Змінюємо імя і виводимо в текстове поле

```
mainthread.setName("cathread");
editResult.setText("Нове ім'я потоку: " + mainthread.getName());
```

Ім'я у головного потоку за замовчуванням **main**, яке ми замінили на **cathread**.

Викличемо інформацію про називу потоку без вказівки методу.

```
Thread mainthread = Thread.currentThread();
textViewInfo.setText("Текущий поток: " + mainthread);
```

У цьому випадку можна побачити рядок **Thread [main, 5, main]** - ім'я потоку, його пріоритет і ім'я його групи.

Створення потоку з інтерфейсом Runnable

Для створення нового потоку потрібно реалізувати інтерфейс **Runnable**. Ви можете створити потік з будь-якого об'єкта, що реалізує інтерфейс **Runnable** і оголосити метод **run ()**.

Усередині методу **run ()** ви розміщуюте код для нового потоку. Цей потік завершиться, коли метод поверне управління.

Коли ви оголосите новий клас з інтерфейсом **Runnable**, вам потрібно використовувати конструктор:

Thread(Runnable об'єкт_потока, String ім'я_потока)

У першому параметрі вказується екземпляр класу, що реалізовує інтерфейс. Він визначає, де почнеться виконання потоку. У другому параметрі передається ім'я потоку.

Після створення нового потоку, його потрібно запустити за допомогою методу **start ()**, який, по суті, виконує виклик методу **run ()**.

Створимо новий потік всередині навчального проекту у вигляді вкладеного класу і запустимо його.

Ключове слово synchronized - синхронізовані методи

Метод може мати модифікатор **synchronized**. Коли потік знаходиться всередині синхронізованого методу, всі інші потоки, які намагаються викликати його в тому ж примірнику, повинні очікувати. Це дозволяє виключити плутанину, коли кілька потоків намагаються викликати метод.

```
synchronized void meow(String msg);
```

Крім того, ключове слово synchronized можна використовувати в якості оператора. Ви можете уклсти в блок synchronized виклики методів якого-небудь класу:

```
synchronized(об'єкт) {  
    // операторы, требующие синхронизации  
}
```

Домашня робота :

Створити потік. Створити клас **MyThread** наслідувати його від класу **Thread**. Перевизначити метод **run()**, так щоб можна було вивести введену з консолі кількість чисел **Фібоначі**. Для цього використовуємо в методі **run()** клас **Scanner**. Тобто ми вводимо з консолі скільки ми хочемо бачити чисел **Фібоначі** і ми їх виводимо з затримкою в 1 секунду.

Створити потік. Створити клас **RunnableThread** імплементувати його від інтерфейсу **Runnable**. Метод **run()** перевизначити таким чином, щоб числа **Фібоначі** виводились в зворотньому порядку. Тобто Спершу ввели через **Scanner** скільки ми хочемо бачити чисел **Фібоначі**, потім з затримкою в 1 секунду вивести їх від останнього до першого значення.

Перший і другий потоки повинні виводити в один рядок числа.

Вигляд на консолі :

Потік Thread : 1 1 2 3 5 ...

Потік Runnable: ... 5 3 2 1 1

Заняття 19

1. Що таке рефлексія?

Рефлексія (від пізньолат. *Reflexio* - звернення назад) - це механізм дослідження даних про програму під час її виконання. Рефлексія дозволяє досліджувати інформацію про поля, методах і конструкторах класів. Можна також виконувати операції над полями і методами які досліджуються. Рефлексія в Java здійснюється за допомогою **Java Reflection API**. Цей інтерфейс API складається з класів пакетів **java.lang** і **java.lang.reflect**. За допомогою інтерфейсу **Java Reflection API (application program interface)** можна робити наступне:

2. Що дозволяє зробити рефлексія?

- Визначити клас об'єкта.
- Отримати інформацію про модифікатори класу, полях, методах, конструкторах і суперклас.
- З'ясувати, які константи і методи належать інтерфейсу.
- Створити екземпляр класу, ім'я якого невідоме до моменту виконання програми.
- Отримати і встановити значення властивості об'єкта.
- Викликати метод об'єкта.
- Створити новий масив, розмір і тип компонентів якого невідомі до моменту виконання програм.

Домашня робота: Створити клас , описати його. Створити в мейн його екземпляр, дослідити всю інформацію про цього.

Заняття 20

1. Розказати про параметризовані класи . Пояснити призначення?

Узагальнене програмування - це такий підхід до опису даних і алгоритмів, який дозволяє їх використовувати з різними типами даних без зміни їх опису. Спочатку вони вводились для строгої типізації колекцій, проте потім отримали і інше призначення.

Параметризовані класи- це класи які дають можливість їх використовувати багато разів для об'єктів найрізноманітніших типів.

Приклад:

```
class DroidsCarrier<T>{
    private T droid;
    public void loadDroid(T droid){
        this.droid = droid;
    }
    public T unloadDroid(){
        return this.droid;
    }
}
```

2. Повторити Set і Map?

HashSet - колекція, що не дозволяє зберігати однакові об'єкти (як і будь Set). HashSet інкапсулює в собі об'єкт HashMap (тобто використовує для зберігання хеш-таблицю).

Як більшість читачів, ймовірно, знають, хеш-таблиця зберігає інформацію, використовуючи так званий механізм хешировання, в якому вміст ключа використовується для визначення унікального значення, званого хеш-кодом. Цей хеш-код потім застосовується в якості індексу, з яким асоціюються дані, доступні з цього ключу. Перетворення ключа в хеш-код виконується автоматично - ви ніколи не побачите самого хеш-коду. Також ваш код не може напряму індексувати хеш-таблицю. Вигода від хешировання полягає в тому, що воно забезпечує константне час виконання методів add (), contains (), remove () і size (), навіть для великих наборів.

Якщо Ви хочете використовувати HashSet для зберігання об'єктів СВОЇХ класів, то ви ПОВИННІ перевизначити методи hashCode () і equals (), інакше два логічно-однакових об'єкта будуть вважатися різними, так як при додаванні елемента в колекцію буде викликатися метод hashCode () класу Object (який швидше-всього поверне різний хеш-код для ваших об'єктів).

Важливо відзначити, що клас HashSet не гарантує впорядкованості елементів, оскільки процес хешировання сам по собі звичайно не породжує сортованих наборів. Якщо вам потрібні сортовані набори, то кращим вибором може бути інший тип колекцій, такий як клас TreeSet.

LinkedHashSet - підтримує зв'язний список елементів набору в тому порядку, в якому вони вставлялися. Це дозволяє організувати впорядковану ітерацію вставки в набір. Тобто, коли йде перебір об'єкта класу LinkedHashSet із застосуванням ітератора, елементи витягаються в тому порядку, в якому вони були додані.

TreeSet - колекція, яка зберігає свої елементи у вигляді упорядкованого за значеннями дерева. TreeSet інкапсулює в собі TreeMap, який в свою чергу використовує збалансоване бінарне червоно-чорне дерево для зберігання елементів. TreeSet хороший тим, що для операцій add, remove і contains потрібно гарантований час $\log(n)$.

PriorityQueue - єдина пряма реалізація інтерфейсу Queue (не рахуючи LinkedList, який більше є списком, ніж чергою).

Ця черга впорядковує елементи або за їх натуральному порядку (використовуючи інтерфейс Comparable), або за допомогою інтерфейсу Comparator, отриманому в конструкторі. Інтерфейс Map співвідносить унікальні ключі зі значеннями. Ключ - це об'єкт, який ви використовуєте для подальшого вилучення даних. Задаючи ключ і значення, ви можете поміщати значення в об'єкт карти. Після того як це значення збережено, ви можете отримати його по ключу. Інтерфейс Map - це узагальнений інтерфейс, оголошений так, як показано нижче.

interface Map<K, V> Тут K вказує тип ключів, а V - тип збережених значень.

Ієархія класів дуже схожа на ієархію Set'a:

HashMap - заснований на хеш-таблицях, реалізує інтерфейс Map (що передбачає зберігання даних у вигляді пар ключ / значення). Ключі і значення можуть бути будь-яких типів, у тому числі і null. Данна реалізація не дає гарантії щодо порядку елементів з плинном часу.

LinkedHashMap - розширює клас HashMap. Він створює зв'язний список елементів у карті, розташованих в тому порядку, в якому вони вставлялися. Це дозволяє організувати перебір карти в порядку вставки. Тобто, коли відбувається ітерація по колекційному поданням об'єкта класу LinkedHashMap, елементи будуть повертатися в тому порядку, в якому вони вставлялися. Ви також можете створити об'єкт класу LinkedHashMap, який повертає свої елементи в тому порядку, в якому до них востаннє здійснювався доступ.

TreeMap - розширює клас AbstractMap і реалізує інтерфейс NavigableMap. Він створює колекцію, яка для зберігання елементів застосовує дерево. Об'єкти зберігаються в відсортованому порядку за зростанням. Час доступу та вилучення елементів досить мало, що робить клас TreeMap близьким вибором для зберігання великих обсягів відсортованої інформації, яка повинна бути швидко знайдена.

WeakHashMap - колекція, яка використовує слабкі посилання для ключів (а не значень). Слабка посилання (англ. Weak reference) - специфічний вид посилань на динамічно створювані об'єкти в системах із збіркою сміття. Відрізняється від звичайних посилань тим, що не враховується складальником сміття при виявленні об'єктів, що підлягають видаленню. Посилання, які не є слабкими, також іноді називають «сильними».

Домашня робота:

Створити клас MyEntry<K,V>. Описати в ньому дженеріками – поля, гетери/сетери, toString. Створити клас Map<K,V>. Реалізвати в даному класі методи, які б:

- Додавали новий об'єкт в мапу
- Видаляли об'єкт мапи за ключем(тобто немає ні ключа ні значення)
- Видаляли об'єкт мапи за значенням(значенню об'єкта присвоюється нульова спилка, при цьому зберігається ключ)
- Виводили на екран Set ключів
- Виводили на екран List значень
- Виводили на екран цілу мапу

Заняття 21 (Тест)

Домашня робота : Завершити завдання що не було завершено на занятті.

Заняття 22-23

Робота над Core Project.

Даний проект має такі сутності: Days, Time, Movie, Seance, Schedule, Cinema.

enum Days:

- прописати дні тижня;

Time:

- int min, int hour;
- передбачити межі для цих полів (для min 0..60, для hour 0..24);

Movie:

- String title, Time duration;

Seance:

- Movie movie, Time startTime, Time endTime;
- в конструктор має надходити параметрами значення для перших двох полів, третє поле повинне обчислюватись (start + duration);

Schedule:

- TreeSet <Seance> (в полі пишемо Set <Seance>, а в конструкторі вже =new TreeSet <Seance>());
- методи: addSeance (Seance), removeSeance (Seance);

Cinema:

- TreeMap<Days, Schedule>, Time open, Time close;
- врахувати час відкриття і закриття при формуванні сеансів!
- методи: addMovie (Movie, Time...time) (додає фільм і зразу ж набір сеансів), addSeance (Seance), removeMovie(Movie) (повністю видаляє фільм з розкладу), removeSeance (Seance, String) (видаляє конкретний сеанс фільму в конкретний день, який задається параметром String).

Main class:

- створення об'єкту Cinema;
- реалізовує меню, в якому виконується весь функціонал Cinema.

Для кожного класу зробити адекватний `toString`, щоб все було читабельно і доступно.

Супроводжуючі повідомлення і тому подібне. Там де потрібно, зробити `compareTo()`. Маєте якісь власні ідеї для розробки - будь-ласка. Це моделювання роботи кінотеатру, тому все що наблизить програму до реальності вітається.

Домашня робота : Попрацювати над проектом .

Заняття 24

1. Повторення всього матеріалу.
2. Моделюємо співбесіду на посаду Junior Java Developer. (В межах Java Core)

Список питань :

1. Назвіть основні принципи ООП ?
2. Що таке інкапсуляція?
3. Що таке Поліморфізм ? Які прояви поліморфізму в Java?
4. Що таке налідування?
5. Опишіть модифікатори доступу в Java?
Чим абстрактний клас відрізняється від інтерфейсу? У яких випадках Ви б використовували абстрактний клас, а в яких інтерфейс?
6. Чи може об'єкт отримати доступ до private-змінної класу? Якщо, так, то яким чином?
7. Які існують типи вкладених класів? Для чого вони використовуються? Наведіть приклад використання вкладеного статичного класу.
8. Чи може статичний метод бути перевизначений?
Опишіть правила правила перевизначення методів класу (overriding). Чи можна змінювати модифікатор доступу методу, якщо так то яким чином? Чи можна змінювати повертається тип, якщо так, то як? Чи можна змінювати тип переданих параметрів?
9. Що таке Generics?
10. Яким чином передаються змінні в методи, за значенням або за посиланням?
11. Які методи є у класу Object? Які методи можна перевизначати, а які ні?
12. Правила перевизначення методу Object.equals () .
13. Що таке конструктор за замовчуванням?
14. Чим відрізняються слова final, finally і finalize?
15. Опишіть ієархію винятків?
16. Що таке checked і unchecked Exception?
17. Конструкція try/catch/finally?
18. Назвіть основні властивості бібліотеки io?
19. Чим відрізняються байтові потоки від символьних потоків?
20. Чим відрізняється StringBuilder від StringBuffer?
21. Яким чином можна створити потік?
22. Чим відрізняється ArrayList від LinkedList? У яких випадках краще використовувати перший, а в яких другий?
23. Що швидше працює ArrayList або LinkedList?
24. Необхідно додати 1 млн. Елемент, яку структуру ви використовуєте?
25. Як відбувається видалення елементів з ArrayList? Як змінюється в цьому випадку розмір ArrayList?
26. Чим відрізняється HashMap і HashTable?
27. Чим відрізняється ArrayList від Vector ?
28. Як побудована HashMap?
29. Яке початкова кількість кошиків в HashMap?
30. Роль equals і hashCode в HashMap?
31. Максимальне число значень hashCode ()?
32. Як і коли відбувається збільшення кількості кошиків в HashMap?
33. У чому відмінності TreeSet і HashSet?
34. Пристрій TreeSet?
35. Як можна відсортувати колекцію в Java?

Логічні задачки:

1. Чому каналізаційні люки круглі?
2. У закритій кімнаті є 3 лампочки, а в коридорі 3 вимикача. За яку мінімальну відкриття дверей можна визначити який вимикач до якої лампочці ставиться?
3. Як розділити торт на 8 рівних частин трьома розрізами?
4. Серед поля знайдений мертвий чоловік із сірником в руках, слідів немає. Від чого він помер і за яких обставин?
5. Скільки тенісних м'ячів поміститься в автобус?
6. Доктор видав пацієнту 4 таблетки двох видів - по 2 таблетки кожного, які не можна відрізнати за зовнішнім виглядом. Таблетки треба випити за два прийоми: вранці по одній таблетці кожного виду і так само ввечері. Якщо порушити дозування або не прийняти таблетки, то пацієнт помре. Так вийшло, що таблетки перемішалися. Як пройти курс лікування і вижити?
7. Шерлок Холмс побачив жінку з простреленою головою, І швидко підбіг до неї і взяв сумочку, Взяв телефон і подзвонив чоловікові сказав, -ваша дружина мертвa !! Чоловік швидко прийшов. І побачивши свою дружину з простреленою головою сказав: «люба, що з тобою» ?? Коли відвезли дружину і прийшла поліція Шерлок Холмс сказав - Ви винні, тоєсть він звинуватив чоловіка !!! Питання: чому Шерлок так думав ???