Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Кафедра програмної інженерії

КОМПЛЕКС НАВЧАЛЬНО-МЕТОДИЧНОГО ЗАБЕЗПЕЧЕННЯ

навчальної дисципліни
"Основи програмування на Java"

підготовки бакалавра
спеціальності 121 «Інженерія програмного забезпечення»
освітньої програми «Програмна інженерія»

Розробник Д.О. Колесников, доцент кафедри ПІ, к.т.н.

Схвалено на засіданні кафедрі ПІ
Протокол від 31 серпня 2017 р. №1

Харків 2017 р.

ЗМІСТ

Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Кафедра Програмної інженерії

ЗАТВЕРДЖУЮ
Декан факультету ____КН___

_____А.Л.Єрохін
(підпис, прізвище, ініціали)
"____" _____2017_ р.

РОБОЧА ПРОГРАМА НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

Основи програмування на Java
(шифр і назва навчальної дисципліни)

напрям підготовки 6.050103 – "Програмна інженерія"

спеціальність _____
(шифр і назва спеціальності)

освітня програма Програмна інженерія
(назва освітньої програми)

факультет Комп'ютерних наук
(назва факультету)

Харків – 2017 р.

Робоча програма з дисципліни «Основи програмування на Java» для студентів усіх форм навчання першого (бакалаврського) рівня вищої освіти напряму підготовки 6.050103 – "Програмна інженерія".
"__" _____, 2017. – __ с.

Розробники: Колесников Дмитро Олегович, к.т.н., доцент кафедри програмної інженерії.

Робочу програму схвалено на засіданні кафедри Програмної інженерії.

Протокол від “____”_____2017 року № ____

Завідувач кафедри ПІ _____ З.В. Дудар
(підпис)
“_____”_____ 2017 р.

Схвалено методичною комісією факультету комп’ютерних наук

Протокол від “____”_____2017 р. № ___

Голова методичної комісії _____ ___О.Ф. Лановий___
(підпис) (ініціали, прізвище)
“_____”_____ 2017 р.

1 ОПИС НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

<table>
<tr><td rowspan="2">Найменування показників</td><td rowspan="2">Галузь знань, напрям підготовки, освітньо-кваліфікаційний рівень</td><td colspan="2">Характеристика навчальної дисципліни</td></tr>
<tr><td>денна форма навчання</td><td>заочна форма навчання</td></tr>
<tr><td>Кількість кредитів ЄКТС – 5</td><td>Галузь знань:
12 "Інформаційні технології"</td><td colspan="2">Цикл дисциплін професійної і практичної підготовки</td></tr>
<tr><td>Модулів – 1</td><td rowspan="4">Напрям підготовки 6.050103 – "Програмна інженерія"</td><td colspan="2">Рік підготовки</td></tr>
<tr><td>Змістових модулів – 1</td><td>2-й</td><td>3-й</td></tr>
<tr><td>Індивідуальних завдань:
РГЗ та КР</td><td colspan="2">Семестр</td></tr>
<tr><td rowspan="6">Загальна кількість годин – 150</td><td>4-й</td><td>6-й</td></tr>
<tr><td rowspan="5">Освітня програма:
"Програмна інженерія"</td><td colspan="2">Кількість годин</td></tr>
<tr><td>70</td><td>36</td></tr>
<tr><td colspan="2">Аудиторні: 1) лекції, год</td></tr>
<tr><td rowspan="11">Тижневих годин для денної форми навчання (17 тижнів):
аудиторних – 4,1
самостійної роботи студента – 4,7</td><td rowspan="11">Освітньо-кваліфікаційний рівень:
бакалавр</td><td>30</td><td>2</td></tr>
<tr><td colspan="2">2) практичні, год</td></tr>
<tr><td>10</td><td>4</td></tr>
<tr><td colspan="2">3) лабораторні, год</td></tr>
<tr><td>20</td><td>12</td></tr>
<tr><td colspan="2">4) консультації, год</td></tr>
<tr><td>10</td><td>18</td></tr>
<tr><td colspan="2">Самостійна робота, год.</td></tr>
<tr><td>80</td><td>114</td></tr>
<tr><td colspan="2">в тому числі інд. завд., год.</td></tr>
<tr><td></td><td></td></tr>
<tr><td colspan="2">Вид контролю: Залік.</td></tr>
</table>

Примітка.

Співвідношення кількості годин аудиторних занять до загальної кількості годин становить: 47% для денної форми навчання, 24% для заочної форми навчання.

2 МЕТА І ЗАВДАННЯ НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

Мета дисципліни.

Метою навчальної дисципліни "Основи програмування на Java" є формування у студентів теоретичних знань та практичних навичок об'єктно-орієнтованого програмування з використанням платформи Java Standard Edition.

Завдання дисципліни: За результатами вивчення дисципліни студенти повинні:

знати:

- мову програмування Java;
- основні класи ядра платформи JSE;
- принципи об'єктно-орієнтованої розробки застосувань мовою Java;

вміти:

- користуватися принципами об'єктно-орієнтованої розробки для написання застосувань мовою Java за допомогою платформи JSE;
- використовувати інтегровані середовища для розробки застосувань на Java;

володіти:

- теоретичними знаннями щодо розробки застосувань мовою Java;
- практичними навичками розробки застосувань мовою Java.

володіти компетенціями:

загально-професійними, до яких відноситься:

- здатність моделювати різні аспекти системи, для якої створюється програмне забезпечення з урахуванням вимог до його якості, надійності, виробничих характеристик (КЗП.05);
- сучасні уявлення про структуру та архітектуру програмного забезпечення, методи проектування програмного забезпечення (КЗП.07);
- здатність використовувати можливості апаратного забезпечення (КЗП.18);
- здатність використовувати можливості операційних систем (КЗП.19);

спеціалізовано-професійними, до яких відносять:

- здатність використовувати професійно-профільовані знання й уміння в галузі практичного використання комп’ютерних технологій (КСП.04);
- використовувати інтернет-ресурси для рішення експериментальних і практичних завдань у галузі професійної діяльності (КСП.05);
- здатність аргументовано переконувати колег у правильності пропонованого рішення, вміти донести до інших свою позицію (КСП.07);
- дотримання професійної етики програмної інженерії (КСП.08);

інструментальними, до яких відносять:

- дослідницькі навички (КІ.03);

− здатність створення технічної документації до програмного проекту (КІ.04).

Навчальна дисципліна "Основи програмування на Java" базується на вивченні таких дисциплін підготовки бакалаврів за напрямом підготовки 6.050103 – "Програмна інженерія": Основи програмної інженерії; Формальні методи програмної інженерії; Людино-машинна взаємодія; Професійна практика програмної інженерії; Якість програмного забезпечення та тестування.

3 ПРОГРАМА НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

Тема 1. Введення в платформу Java. Кодування.
Тема 2. Лексична трансляція. Типи даних та літерали.
Тема 3. Операції та оператори.
Тема 4. Класи і конструктори. Перетворення між типами.
Тема 5. Абстрактні класи та інтерфейси. Вкладені типи.
Тема 6. Перелічувальний тип.
Тема 7. Generics. Параметризовані класи та методи.
Тема 8. Строкові класи. Інтернаціоналізація.
Тема 9. Регулярні вирази.
Тема 10. Механізм винятків.
Тема 11. Потоки введення / виводу.
Тема 12. Порівняння об'єктів.
Тема 13. Колекції.
Тема 14. Асоціативні контейнери.
Тема 15. Багатопоточне програмування.

4 СТРУКТУРА НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

Назви змістових модулів і тем	Кількість годин											
	денна форма						заочна форма					
	усього	у тому числі					усього	у тому числі				
		лк	пз	лр	Кс	ср		лк	пз	лр	кс	ср
1	2	3	4	5	6	7	8	9	10	11	12	13
Введення в платформу Java. Кодування.	8	2				6	11	2			1	8
Лексична трансляція. Типи даних та літерали.	8	2				6	9				1	8
Операції та оператори.	14	2		4	2	6	14			4	2	8
Класи і конструктори. Перетворення між типами.	16	2	2	4	2	6	16		2	4	2	8
Абстрактні класи та інтерфейси. Вкладені типи.	8	2				6	9				1	8
Перелічувальний тип.	7	2				5	9				1	8

Строкові класи. Інтернаціоналізація.	13	2	2	4		5	15		2	4	1	8
Регулярні вирази.	9	2			2	5	10				2	8
Generics. Параметризовані класи та методи.	9	2	2			5	9				1	8
Механізм винятків.	7	2				5	8				1	7
Потоки введення / виводу.	13	2		4	2	5	8				1	7
Порівняння об'єктів.	9	2	2			5	8				1	7
Колекції.	7	2				5	8				1	7
Асоціативні контейнери.	15	2	2	4	2	5	8				1	7
Багатопоточне програмування.	7	2				5	8				1	7
Загальна кількість	150	30	10	20	10	80	150	2	4	12	18	114

5 ТЕМИ ПРАКТИЧНИХ ЗАНЯТЬ

№	Назва теми	Кількість годин	
		денна	заочна
1	Класи і конструктори. Перетворення між типами.	2	2
2	Строкові класи. Регулярні вирази. Інтернаціоналізація.	2	2
3	Generics. Параметризовані класи та методи.	2	
4	Порівняння об'єктів.	2	
5	Асоціативні контейнери.	2	
	Загальна кількість	10	4

6 ТЕМИ ЛАБОРАТОРНИХ ЗАНЯТЬ

№	Назва теми	Кількість годин	
		денна	заочна
1	Робота з операторами та масивами.	4	4
2	Робота з класами.	4	4
3	Робота зі строками.	4	4
4	Робота з IO.	4	
5	Робота з асоціативними контейнерами.	4	
	Загальна кількість	20	12

7 САМОСТІЙНА РОБОТА

№	Назва теми	Кількість годин	
		денна	заочна
1	Введення в платформу Java. Кодування.	6	8
2	Лексична трансляція. Типи даних та літерали.	6	8
3	Операції та оператори.	6	8
4	Класи і конструктори. Перетворення між типами.	6	8
5	Абстрактні класи та інтерфейси. Вкладені типи.	6	8
6	Перелічувальний тип.	5	8
7	Строкові класи. Інтернаціоналізація.	5	8
8	Регулярні вирази.	5	8

9	Generics. Параметризовані класи та методи.	5	8
10	Механізм винятків.	5	7
11	Потоки введення / виводу.	5	7
12	Порівняння об'єктів.	5	7
13	Колекції.	5	7
14	Асоціативні контейнери.	5	7
15	Багатопоточне програмування.	5	7
	Загальна кількість	80	114

8 ІНДИВІДУАЛЬНІ ЗАВДАННЯ

1. Розрахунково-графічні завдання (РГЗ) не передбачені планом.
2. З дисципліни передбачена контрольна робота для заочної форми навчання.
3. Курсова робота не передбачена планом.

9 МЕТОДИ НАВЧАННЯ

1. Методи організації та здійснення навчально-пізнавальної діяльності за допомогою слайд-лекцій, пояснень на багатьох практичних прикладів.

2. Методи стимулювання й мотивації навчально-пізнавальної діяльності студентів у виконанні власних проектів з практичної реалізації завдань дисципліни.

3. Методи контролю (самоконтролю, корекції) за ефективністю навчально-пізнавальної діяльності студента – це є методи, які спрямовані на самостійну, творчу та пізнавальну діяльність студентів, особливо при створені власних проектів.

4. Універсальні методи поєднують самостійну роботу студентів під час практичних занять з інструктуванням, допомогою викладача, у результаті чого студенти набувають навичок щодо проведення самостійної роботи поза аудиторного навантаження. Крім цього студент має у своєму розпорядженні слайд-лекції, приклади розв’язання задач з роз’ясненнями, які поєднуються в наочно-ілюстративно-практичний комплект матеріалів для навчання.

10 МЕТОДИ КОНТРОЛЮ ТА РЕЙТИНГОВА ОЦІНКА ЗА ДИСЦИПЛІНОЮ

Усне опитування студентів допомагає контролювати не лише знання, а й вербальні вміння, сприяє виправленню мовних помилок. Відтворення студентом раніше вивченого матеріалу сприяє кращому запам'ятовуванню, активному використанню наукових понять, що неможливо без достатнього застосування їх у мові.

Усне опитування може бути індивідуальним і фронтальним. За фронтального опитування студенти відповідають з місця, доповнюючи один одного. Частковим випадком фронтального опитування є групове опитування – 5-6 осіб одночасно. Індивідуальне опитування здійснюється у процесі проведення співбесіди під час практичних занять.

Запитання для усної перевірки знань поділяють на основні, додаткові і допоміжні. Основні запитання передбачають самостійну розгорнуту відповідь (наприклад, запитання щодо змісту лабораторного заняття). Додаткові спрямовані на уточнення того, як студент розуміє певне питання, формулювання, формулу та ін. Допоміжні запитання мають за мету виправлення помилок та неточностей, якщо такі мали місце у відповіді студента. Усі запитання повинні бути логічними, чіткими, зрозумілими, а їх сукупність – послідовна і системна.

Письмовий контроль можна здійснюється у вигляді відповідей на запитання, розв'язання задач під час виконання практичних робіт. Письмові роботи допомагають за короткий час з'ясувати рівень засвоєння матеріалу у великої кількості студентів. Результати письмових робіт можна проаналізувати і з'ясувати деталі і неточності у відповідях та аналізувати їх причини.

Формою письмового контролю рівня знань з дисципліни є комплексна контрольна робота.

Практичний контроль передбачає виявлення вмінь і навичок студентів, що набуті під час практичної діяльності (практичні заняття, робота над власним проектом). Така перевірка дає змогу виявити, на якому рівні студент засвоїв теоретичні основи цих дій.

Як форма підсумкового контролю для дисципліни використовується залік.

10.1 Розподіл балів, які отримують студенти (Кількісні критерії оцінювання)

Вид заняття/ контрольний захід	Оцінка $О_{сем}$	
	Денна	Заочна
Лб № 1	6-10	12-20
Лб № 2	6-10	12-20
Лб № 3	6-10	12-20
Лб № 4	6-10	
Лб № 5	6-10	
Пз № 1	6-10	12-20
Пз № 2	6-10	12-20
Пз № 3	6-10	
Пз № 4	6-10	
Пз № 5	6-10	
Всього за семестр	60-100	60-100

10.2 Якісні критерії оцінювання

Необхідний обсяг знань для одержання позитивної оцінки:

- знати мову програмування Java;
- знати основні класи ядра платформи JSE;

− знати принципи об'єктно-орієнтованої розробки застосувань мовою Java.

Необхідний обсяг умінь для одержання позитивної оцінки:

− вміти користуватися принципами об'єктно-орієнтованої розробки для написання застосувань мовою Java за допомогою платформи JSE;
− використовувати інтегровані середовища для розробки застосувань на Java.

10.3 Критерії оцінювання роботи студента протягом семестру

Як форма підсумкового контролю для дисципліни використовується залік. При цьому виді контролю підсумкова оцінка $Р_п$ обчислюється за формулою $Р_п = О_{сем}$, де – $О_{сем}$ оцінка за семестр в 100 бальній системі.

Критерії оцінювання роботи студента протягом семестру.

Задовільно D, E (60-74). Мати мінімум знань та вмінь. Виконати та захистити усі лабораторні роботи. Виконати всі завдання під час практичних занять.

Добре C (75-89). Твердо засвоїти мінімум знань. Вміти використовувати ці знання при розв'язанні практичних завдань. Виконати та захистити усі лабораторні роботи в строк. Виконати усі пункти завдань до практичних занять в строк.

Відмінно A, B (90-100). Знати усі теми та вільно орієнтуватися у предметній галузі дисципліни. Виконати та захистити усі лабораторні роботи та практичні завдання в строк з найвищою оцінкою. Виконати усі індивідуальні завдання до практичних занять в строк з найвищою оцінкою.

Критерії оцінювання знань та вмінь студента під час заліку:

Задовільно D, E (60-74). Показати необхідний мінімум теоретичних знань. Вміти використовувати ці знання при розв'язанні практичних задач.

Добре C (75-89). Твердо орієнтуватися в теоретичній складовій курсу. Вміти створювати на практиці застосування мовою Java. Використовувати інтегровані середовища для розробки застосувань на Java.

Відмінно A, B (90-100). Показати максимально повні знання в предметній галузі дисципліни. Обґрунтувати рішення практичних задач.

Шкала оцінювання: національна та ЄКТС

Сума балів за всі види навчальної діяльності	Оцінка ЄКТС	Оцінка за національною шкалою	
		для іспиту, курсового проекту (роботи), практики	для заліку
96-100	A	відмінно	зараховано
90-95	B		
75-89	C	добре	
66-74	D	задовільно	
60-65	E		
35-59	FX	незадовільно з можливістю повторного складання	не зараховано з можливістю повторного складання
0-34	F	незадовільно з обов'язковим повторним вивченням дисципліни	не зараховано з обов'язковим повторним вивченням дисципліни

11 МЕТОДИЧНЕ ЗАБЕЗПЕЧЕННЯ ТА РЕКОМЕНДОВАНА ЛІТЕРАТУРА

11.1 Базова література

1. Шилдт Г. Java 8. Полное руководство : пер. с англ. / Г. Шилдт. – 9-е изд. – М.: Вильямс, 2015. – 1376 с.

2. The Java Language Specification. Java SE 8 Edition. [Електронний ресурс] http://docs.oracle.com/javase/specs/jls/se8/jls8.pdf

11.2 Допоміжна література

3. The Java Tutorials. [Електронний ресурс] http://docs.oracle.com/javase/tutorial

4. Хорстманн К., Корнелл Г. Java 2. Библиотека профессионала. Том первый. Основы. - М.: Вильямс, 2014. - 1008 с.

5. Хорстманн К., Корнелл Г. Java 2. Библиотека профессионала. Том второй. Тонкости программирования. - М.: Вильямс, 2014. - 864 с.

11.3 Інформаційні ресурси

6. http://www.oracle.com/technetwork/java/javase/tech/index.html

11.4 Методичне забезпечення

7. Конспект лекцій з дисципліни "Основи програмування на Java" для студентів усіх форм навчання напряму підготовки 6.050103 – "Програмна інженерія" [Електронний ресурс] / ХНУРЕ; Д.О. Колесников – Харків: ХНУРЕ, 2017. – 103 с. Електронна версія (pdf / 537 Kb).

8. Методичні вказівки до практичних робіт з дисципліни "Основи програмування на Java" для студентів усіх форм навчання напряму підготовки 6.050103 – "Програмна інженерія" [Електронний ресурс] / ХНУРЕ; розроб. Д.О. Колесников. – Харків: ХНУРЕ, 2017. – 8 с. Електронна версія (pdf / 333 Kb).

9. Методичні вказівки до самостійної роботи з дисципліни "Основи програмування на Java" для студентів усіх форм навчання напряму підготовки 6.050103 – "Програмна інженерія" [Електронний ресурс] / ХНУРЕ; розроб. Д.О. Колесников. – Харків: ХНУРЕ, 2017. – 20 с. Електронна версія (pdf / 353 Kb).

10. Методичні вказівки до лабораторних робіт з дисципліни "Основи програмування на Java" для студентів усіх форм навчання напряму підготовки 6.050103 – "Програмна інженерія" [Електронний ресурс] / ХНУРЕ; розроб. Д.О. Колесников. – Харків: ХНУРЕ, 2017. – 20 с. Електронна версія (pdf / 445 Kb).

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни
"ОСНОВИ ПРОГРАМУВАННЯ НА JAVA"

для студентів напряму підготовки
6.050103 – "Програмна інженерія"

Електронне видання

Харків 2017

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни
"ОСНОВИ ПРОГРАМУВАННЯ НА JAVA"

для студентів напряму підготовки
6.050103 – "Програмна інженерія"

Електронне видання

ЗАТВЕРДЖЕНО
кафедрою ПІ.
Протокол № ___ від _____ р.

Харків 2017

Конспект лекцій з дисципліни "Основи програмування на Java" усіх форм навчання напряму підготовки 6.050103 – "Програмна інженерія" [Електронне видання] / Упоряд.: Д.О. Колесников. – Харків: ХНУРЕ, 2017. – 103 с.

Упорядник: Д.О. Колесников, доц. кафедри ПІ.

ЗМІСТ

КОНСПЕКТ ЛЕКЦІЙ
з дисципліни "Основи програмування на Java"

1 КОДУВАННЯ

1.1 Кодування

Кодування - це відповідність між символами і числами. Кожен символ кодування має фіксований унікальний числовий код. Кодування можна представити у вигляді таблиці:

символ	код
A	65
B	66
C	67
...	...

Як приклади кодувань можна вказати наступні:

- ASCII - складова частина більшості кодувань, що містить 128 символів;
- Unicode - містить більше мільйона символів, для запису символів застосовують різні види UTF;
- Windows-1251 або Cp1251 - однобайтном кодування, містить кирилицю, є кодуванням за замовчуванням Windows російської локалізації;
- Cp866 або DOS-кодування - однобайтном кодування, містить кирилицю, є кодуванням за замовчуванням консолі Windows російської локалізації;
- KOI8 - однобайтном кодування, містить кирилицю (підвиди: KOI8-R або Cp20866 - містить російський алфавіт, KOI8-U або Cp21866 - містить український алфавіт);
- ISO-8859-1 або Latin-1 або Cp819 - перші 256 символів кодування Unicode.

Практично кожна операційна система має т.зв. кодування за замовчуванням (КПУ).

1.2 Кодування ASCII

Кодування ASCII включає в себе керуючі символи, знаки пунктуації, десяткові цифри, латинський алфавіт. Коди символів ASCII лежать в діапазоні від 0 до 127 включно. Практично всі поширені кодування включають в себе ASCII кодування складовою частиною в початкових 128 позиціях.

1.3 Керуючі символи ASCII

Керуючі символи - це символи, які не мають графічного представлення. Їх використовують для управління пристроями. Приклад керуючих символів: табуляція (горизонтальна, вертикальна), повернення каретки, звуковий сигнал, новий рядок, переклад сторінки.

Важливими керуючими символами, які часто використовують є:

- повернення каретки (позначення: '\r', CR) - символ з кодом 13 (D в шістнадцятковій системі числення). Англійська назва - carriage return. Висновок символу CR тягне переміщення курсора на початок поточного рядка;
- переклад рядка (позначення: '\n', LF) - символ з кодом 10 (A в шістнадцятковій системі числення). Англійська назва - line feed. Висновок символу LF тягне переміщення курсору на початок наступного рядка.

1.4 Unicode

Unicode - це стандарт кодування символів. Загалом складається з двох частин:

- кодування Unicode (містить понад мільйон символів);
- формат перетворення Unicode (UTF - Unicode Transformation Format).

Кожен символ Unicode має фіксований числовий код, т.зв. кодову точку (code point), у вигляді невід'ємного цілого числа.

Для позначення символів кодування Unicode використовують наступну нотацію:

- U + xxxx - для символів з кодовими точками з діапазону [0, FFFF];
- U + xxxxx - для символів з кодовими точками з діапазону [1 0000 F FFFF];
- U + xxxxxx - для символів з кодовими точками з діапазону [10 0000 10 FFFF].

Діапазони вказані в шістнадцятковій системі числення.

Кількість символів в кодуванні Unicode залежить від версії стандарту. Максимальна кодова точка кодування Unicode - 10FFFF, проте загальна кількість символів менше цього значення, тому що деяким кодами символи у відповідність не поставлені.

1.5 Діапазони кодування Unicode

- [U + 0000 U + 007F] - збігається з ASCII;
- [U + 0000 U + FFFF] - BMP (Basic Multilingual Plane), базова багатомовна площину (містить найбільш часто використовувані символи);
- [U + 10000, U + 10FFFF] - додаткові символи (supplimentary characters).

1.6 UTF

UTF визначає, як кодові точки будуть представлені байтами. Кожен вид UTF робить це по своєму. З формальної точки зору UTF не є кодуванням. Співвідношення між UTF і кодуванням Unicode схематично можна показати наступним чином:

символ <== Unicode кодування ==> Код <== UTF ==> Байти

Існує принаймні три види UTF:

UTF	Кількість байт на символ
UTF-8	від 1 до 6 байт, для запису символів ASCII використовує один байт
UTF-16	2 байта для символів [U + 0000 U + FFFF] 4 байта для символів [U + 10000, U + 10FFFF]
UTF-32	4 байта

Також UTF-16 і UTF-32 мають два варіанти, пов'язаних з прямим або зворотним порядком запису байт: UTF-16BE, UTF-16LE, UTF-32BE, UTF-32LE.

1.7 Порядок байт

Там, де для запису числа потрібно більше одного байта, важливим є порядок байт. Для UTF використовують два порядки - прямий порядок байт в слові BE (Big Endian) і зворотний порядок байт в слові LE (Little Endian). Під словом розуміють два байта.

При використанні прямого порядку BE старший (більш значимий) байт в слові знаходиться попереду молодшого (менш значимого) байта. При використанні зворотного порядку LE молодший (менш значимий) байт в слові розташований попереду старшого (більш значущого) байта.

1.8 Мітка порядку байт

Мітка порядку байт (BOM - Byte Order Mark) представляє з себе сукупність байт, яка розташована в самому початку потоку інформації і яка вказує в якому порядку записані наступні байти. BOM залежить від виду UTF:

UTF	BOM
UTF-16BE	FE FF
UTF-16LE	FF FE
UTF-32BE	00 00 FE FF

UTF-32LE	FF FE 00 00
UTF-8	EF BB BF

BOM для UTF-8 відіграє ідентифікує роль (тобто, в даному випадку BOM не вказує порядок байт, а вказує, що подальша інформація представлена в форматі UTF-8).

Стандарт Unicode визначає використання мітки порядку байт як не обов'язкова, тобто, BOM може бути відсутнім. За замовчуванням (в разі відсутності BOM) порядок байт приймають прямим - BE.

1.9 Діапазони сурогатних заменітілей UTF-16

Для уявлення кожного додаткового символу з діапазону [U + 10000, U + 10FFFF] UTF-16 використовує два сурогатних символу, т.зв. сурогатні замінники.

Діапазони сурогатний замінників:

- [U + D800, U + DBFF] - верхній;
- [U + DC00, U + DFFF] - нижній.

Всього для запису додаткових символів потрібно 4 байта - по два байта на кожен сурогатний символ.

2 ЛЕКСИЧНА ТРАНСЛЯЦІЯ

2.1 Unicode escape послідовність Java

Unicode escape послідовність Java вдає із себе вираз виду \uXXXX, де XXXX - шістнадцятковий код символу в кодуванні Unicode. Регістр шістнадцятирічних цифр не має значення, але буква u повинна бути в нижньому регістрі. С допомогою таких послідовностей можна уявити в вихідному коді Java будь-який символ з BMP Unicode.

Для запису додаткових символів Unicode використовують дві поспіль йдуть escape послідовності, в яких записані коди відповідних сурогатних замінників: U + 1D120 ==> \uD834 \uDD20.

2.2 Лексична трансляція коду програми

В результаті трьох переглядів вихідного коду лексичний транслятор, вбудований в компілятор Java, здійснює послідовно наступні три маніпуляції:

1) підстановка \uXXXX ==> символ Unicode з кодовою точкою XXXX (U + XXXX);

2) визначення вхідних Unicode символів і обмежувачів рядків (тобто розпізнавання рядків);

3) визначення вхідних елементів (пробільні символи, коментарі, лексеми).

2.3 Обмежувачі рядків

Нижче перераховані обмежувачі рядків в Java:

- символ U + 000A, він же ASCII-символ LF (переклад рядка);
- символ U + 000D, він же ASCII-символ CR (повернення каретки);
- впорядкована послідовність символів U + 000D і U + 000A.

2.4 Вхідні елементи мови Java

Нижче перераховані вхідні елементи мови Java:

- пробільні символи;
- коментарі;
- лексеми.

Лексеми відокремлені один від одного пробільними символами або коментарями:

```
int/* коментар розділяє лексеми */x;
```

2.5 Пробільні символи

Пробільні символи служать для поділу лексем:

- пробіл (SP);
- горизонтальна табуляція (HT);
- переклад сторінки (FF);
- обмежувачі рядків (\u000A, \u000D, \u000D \u000A).

2.6 Коментарі

Існує два види коментарів:

- однорядковий: // текст
- багатостроковий: / * текст * /

Складний коментар виду / ** документація * / також називають коментарем Документатор. Записана в такого виду коментарі інформація в подальшому може бути використана спеціальними утилітами, які складають документацію до коду (таким чином, коментарі до коду розташовані в самому коді).

2.7 Лексеми

Всього існує п'ять видів лексем (в дужках вказані діапазони символів з яких можуть бути складені відповідні лексеми):

Ідентифікатори (Unicode)

Літерали (Unicode)

Ключові слова (ASCII)

Роздільники (ASCII)

Знаки операцій (ASCII)

2.8 Ідентифікатори

Ідентифікатори використовують для іменування:

- типів (класів, інтерфейсів);
- пакетів;
- методів;
- полів;
- локальних змінних.

Ідентифікатори вдають із себе послідовність літер і цифр мови Java. На першому місці в послідовності повинна бути буква.

Буквою в Java називають символ, для якого метод

java.lang.Character.isJavaIdentifierStart

повертає значення true.

Приклади букв:

- символи латинського, російського, китайського алфавітів;
- символи підкреслення _ і долара $.

Буквою або цифрою в Java називають символ, для якого метод

java.lang.Character.isJavaIdentifierPart

повертає значення true.

Приклади символів, які є буквою або цифрою:

- символи латинського, російського, китайського алфавітів;
- символи підкреслення _ і долара $;
- цифри від 0 до 9 (діапазон [U + 0030; U + 0039]).

Рекомендація. Пишіть ідентифікатори англійською мовою, загальноприйнятим є іменування ідентифікаторів таким чином, щоб за назвою було зрозуміло, що він представляє (не використовуйте угорську нотацію, абревіатури, транслітерацію, більш детально з ім'ям ідентифікаторів можна познайомитися в документі [1], а також книзі [2]).

2.9 Ключові слова

У мові Java версії 7 існує 50 ключових слів:

`abstract`	`assert`	`boolean`	`break`	`byte`
`case`	`catch`	`char`	`class`	`const`
`continue`	`default`	`do`	`double`	`else`
`enum`	`extends`	`final`	`finally`	`float`
`for`	`goto`	`if`	`implements`	`import`
`instanceof`	`int`	`interface`	`long`	`native`
`new`	`package`	`private`	`protected`	`public`
`return`	`short`	`static`	`strictfp`	`super`
`switch`	`synchronized`	`this`	`throw`	`throws`
`transient`	`try`	`void`	`volatile`	`while`

Деякі категорії ключових слів перераховані нижче:

- примітивні типи даних:
  - числа
    - цілі: byte, short, int, long, char
    - речові: float, double
  - логічний тип: boolean
- модифікатори рівня доступу: public, protected, private
- ключові слова, використовувані в операторах вибору:

- умовний оператор: if, else
- оператор багатозначно вибору: switch, case, default

- оператори циклу: for, while, do
- використовувані при роботі з винятками:
  - оператор викиду: throw
  - секція вказівки потенційно викидаються винятків: throws
  - блок роботи з винятками: try, catch, finally
- зарезервовані, але не використовуються: goto, const
- оператори визначення нових типів: class, enum, interface

2.10 Роздільники

Роздільниками є такі символи:

- дужки
  - квадратні []
  - круглі ()
  - фігурні {}
- крапка .
- кома,
- крапка з комою ;

2.11 Операції

В Java 7 існує 25 знаків операцій:

знак	арность	операція
+	2	складання цілих чисел
	2	складання дійсних чисел
	2	з'єднання з рядком (будь-якого об'єкта або примітиву)
	1	унарний плюс

знак	арность	операція
-	2	віднімання цілих чисел
	2	віднімання дійсних чисел
	1	унарний мінус

знак	арность	операція
*	2	множення цілих чисел
	2	множення дійсних чисел

знак	арность	операція
/	2	розподіл цілих чисел
	2	розподіл дійсних чисел

знак	арность	операція
%	2	залишок від ділення цілих чисел
	2	залишок від ділення дійсних чисел

знак	арность	оператор
++	1	інкремент в префиксной формі записи
	1	інкремент в постфіксной формі записи

знак	арность	оператор
–	1	декремент в префиксной формі записи
	1	декремент в постфіксной формі записи

знак	арность	оператор
>	2	порівняння чисел (більше)
> =	2	порівняння чисел (більше або дорівнює)
<	2	порівняння чисел (менше)
<=	2	порівняння чисел (менше або дорівнює)

знак	арность	оператор
==	2	рівності чисел (одно)
==	2	рівності boolean (так само)
==	2	рівності об'єктів (так само)

знак	арность	оператор
! =	2	нерівності чисел
! =	2	нерівності boolean
! =	2	нерівності об'єктів

знак	арность	операція
&	2	логічна операція "І"
	2	бітова операція "І"

знак	арность	операція
|	2	логічна операція "АБО"
	2	бітова операція "АБО"

знак	арность	операція
^	2	логічна операція "Що виключає АБО"
	2	бітова операція "Що виключає АБО"

знак	арность	операція
&&	2	логічна операція "І по короткій схемі"

знак	арность	операція

`||`	2	логічна операція "АБО по короткій схемі"

знак	арность	операція
!	1	логічне заперечення

знак	арность	операція
~	1	бітове доповнення

знак	арность	оператор
>>	2	зсув вправо
<<	2	зсув вліво
>>>	2	зсув вправо зі збереженням знака

знак	арность	оператор
=	2	присвоювання
?:	3	оператор вибору

Також існує складовою оператор присвоювання виду

```
X op = Y
```

який представляє з себе скорочений запис виразу

```
X = (type of X) (X op Y)
```

вичерпний список складових операторів:

`+ =`	`- =`	`* =`	`/ =`	`% =`
`& =`	`| =`	`^ =`		
`<< =`	`>> =`	`>>> =`		

У наведених вище операціях в якості операндів можуть бути використані об'єкти класів оболонок.

3 ЛІТЕРАЛИ

Літерали - це уявлення в вихідному коді програми:

- значень примітивних типів
  - int, long (цілі літерали);
  - float, double (речові літерали);
  - char (символьні літерали);
  - boolean (булеві літерали);
- значень типу String (рядкові літерали);
- значень типу масив (масиви константи);
- null - літерал нул-типу (null-літерал).

Зауваження. Параметризрвані екземпляри Class <Type> часто також називають літералами типу Type.

3.1 Числові літерали

Числові літерали - константи типів:

- int, long (цілі літерали)
- float, double (речові літерали)

У записі числових літералів допустимо використовувати знак підкреслення для поділу розрядів (скільки завгодно знаків _ може знаходиться між цифрами:

```
10_000    0_7777    1______2_3E1_2
```

Якщо числовий літерал (цілий або дійсний) передує знак + або -, то знак "+/-" не входить до складу литерала, наприклад нижче представлені два арифметичних вирази (але не литерала):

```
-34  +3
```

3.2 Цілі числові літерали

Цілі літерали можуть бути записані за допомогою однієї з чотирьох систем числення: десятковій, шістнадцятковій, вісімковій і бінарної.

Якщо в кінці цілого литерала варто суфікс L або l, то тип литерала long, якщо суфікс відсутній, то тип литерала int.

Негативні числа можуть бути представлені тільки за допомогою бінарних, вісімкових або шістнадцяткових литералов. Три наступних литерала представляють -1:

- 0b11111111_11111111_11111111_11111111 (бінарний цілий літерал)
- 037_777_777_777 (восьмеричний цілий літерал)

• 0xFF_FF_FF_FF (шістнадцятковий цілий літерал)

Десяткові літерали не можуть представляти негативні числа, тільки невід'ємні (позитивні або нуль).

3.2.1. Десяткові цілі літерали

структура:

• мінімум одна десяткова цифра;

• необов'язковий суфікс L / l;

• якщо цифр більше ніж одна, то перша не може бути нулем.

Приклади: 0; 123; 0L; 0l; 72L; 5.

Діапазони десяткових цілих литералов:

• для int: [0, 2 ^ 31]

• для long: [0, 2 ^ 63]

При цьому максимальні по величині літерали можуть бути використані тільки з унарною операцією зміни знака:

• int: 2 ^ 31 = -2147483648

• long: 2 ^ 63 = -92233720368547758O8L

Без мінуса:

• int: 2 ^ 31 - 1 = 2147483647

• long: 2 ^ 63 - 1 = 9223372036854775807L

3.2.2. Шістнадцятиричні цілі літерали

структура:

• обов'язковий ознака 0x або 0X;

• мінімум одна Шістнадцяткова цифра (0 - 9, a - f, A - F);

• опціональний суфікс L / l.

Приклади: 0xABL; 0X0; 0x123L; 0X123.

3.2.3. Вісімкові цілі літерали

структура:

• обов'язковий ознака вісімкового литерала 0;

• мінімум одна восьмерична цифра (0-7);

• опціональний суфікс L / l.

Приклади: 00; 00000L; 017; 0777L; 0123.

3.2.4. Бінарні цілі літерали

структура:

• обов'язковий ознака 0b або 0B;

• мінімум одна цифра з безлічі {0, 1};

• опціональний суфікс L / l.

Приклади: 0b101L; 0B00000.

3.3 Речові числові літерали

Речові літерали можуть бути записані за допомогою десяткового або шістнадцятковій систем числення.

Якщо в кінці речового литерала варто суфікс F або f, то тип литерала float, якщо стоїть суфікс D або d або ж суфікс відсутній, то тип литерала double.

3.3.1. Десяткові речові літерали

Загальний вигляд (у дужках - приклад литерала):

[Цифри] [точка] [цифри] [десяткова експонента] [суфікс] (1.2E-3D)

При цьому: цифри - десяткові (0-9), а суфікс - один з D, d, F, f. Всього є чотири варіанти структури (інші компоненти можуть як бути присутніми, так і бути відсутнім):

1) [Цифри] [точка] (12 .; 1.2; 1.e + 2; 1.2f);

2) [Точка] [цифри] (.12; .1; .1E2; 1.23);

3) [Цифри] [десяткова експонента] (12E3; 1e-2d; 1.2E + 3);

4) [Цифри] [суфікс] (1f; 12D; .1D, 1E2D).

При записи десяткових речових літералів може бути використана десяткова експонента (це т.зв. літерали записані в науковій формі), яка має таку загальну структуру:

• обов'язковий ознака десяткової експоненти E або e;

• необов'язковий знак експоненти + або -;

• мінімум одна десяткова цифра.

Приклади (в дужках значення експоненти):

• E1 (10 ^ 1 = 10);

• e + 1 (10 ^ + 1 = 10);

• E-123 (10 ^ -123).

Наприклад, запис 123E-45 означає число 123 * 10 ^ -45.

3.3.2. Шістнадцятиричні речові літерали

Загальний вигляд (у дужках - приклад литерала):

[0X або 0x] [ц] [точка] [ц] [бін. експ.] [суфікс] (0X12.34P-5D)

де 'ц' - цифра; 'Бін. експ. ' - бінарна експонента.

При цьому:

• бінарна експонента є необхідною;

• цифри - шістнадцяткові (0-9, AF, af);

• суфікс - один з D, d, F, f.

Приклади: 0x24P1; 0x1.2p-3F

При записи шістнадцятирічних речових літералів завжди потрібно вказувати бінарну експоненту, яка має таку загальну структуру:

• повинен бути ознака бінарної експоненти P або p;

- знак експоненти + або - є не обов'язковим;
- повинна бути мінімум одна десяткова цифра.

Приклади: p1; p + 1; P-99.

Як буде обчислено значення бінарної експоненти показано на прикладі:

```
23.4P3 = (2 * 161 + 3 * 160 + 4 * 16-1) * 23
```

3.4 Булеві літерали і літерал Null-тип

Булеві літерали - це константи типу boolean, все їх безліч вичерпується двома значеннями: true, false.

null літерал - це константа спеціального Null-типу, який сам по собі не має імені (тобто не можна оголосити змінну такого типу) і представлений всього лише одним значенням - null.

3.5 Символьні літерали

Символьний літерал - це символ Unicode з діапазону [U + 0000 U + FFFF], укладений в одинарні лапки '(U + 0027), за винятком:

- одинарної лапки '(U + 0027);
- зворотного слеша \(U + 005C);
- Unicode escape послідовностей \u000A, \u000D.

Символьні літерали мають тип char, а отже, з огляду на те, що char є числовим типом, представляють деякий число з діапазону [0, FFFF].

Приклади: 'a'; 'T'; '\U0065'; '\77'; '\\'

3.6 Рядкові літерали

Рядкові літерали - це нуль або більше символів Unicode (допустимі символи з усього діапазону Unicode), укладені в подвійні лапки "(U + 0022), за винятком:

- подвійної лапки "(U + 0022);
- зворотного слеша \(U + 005C);
- Unicode escape послідовностей \u000A, \u000D.

Рядкові літерали мають тип String.

Приклади: ""; "Ab \" c ";" \u0065bc \123 "

Довгий строковий літерал може бути записаний за допомогою оператора конкатенації рядків +, результат конкатенації - строковий літерал. Конкатенація двох строкових літералів - вираз, а не строковий літерал, однак, результат такого виразу буде вирахувано на етапі компіляції.

3.6.1. Подання символів в літералах

У символьних і строкових літералах символ може бути представлений у вигляді:

- знака символу;
- Unicode escape послідовності \uXXXX (крім \u000D і \u000A)
  - одним - для символів з BMP Unicode;
  - двома - для додаткових символів.

- восьмеричної escape послідовністю (символи ISO-8859-1):
  - \X (діапазон [\0 .. \7])
  - \XX (діапазон [\00 .. \77])
  - \XXX (діапазон [\000 .. \377])
- символьної escape послідовністю (тільки ці 8 символів):
  - \\\t \r \n \'\"\b \f

3.6.2. Символьні escape послідовності

Символьні escape послідовності представляють деякі символи мнемонічною записом з використанням символу \, вичерпний список послідовностей наведено в таблиці:

- \t U + 0009 горизонтальна табуляція
- \n U + 000A переклад рядка
- \r U + 000D повернення каретки
- \f U + 000C переклад сторінки
- \' U + 0027 одинарна лапка
- \" U + 0022 лапки
- \\ U + 005C зворотний слеш
- \b U + 0008 забій (backspace)

3.6.3. Вісімкові escape послідовності

Вісімкові escape послідовності можуть представляти символи з діапазону від 0 до 255 (тобто Latin-1). Загальна структура наступна:

- \A - символ з кодом 0A;
- \AB - символ з кодом 0AB;
- \ZAB - символ з кодом 0ZAB.

При цьому:

- Z - цифра з безлічі [0, 3];
- A, B - восьмеричні цифри [0, 7].

Приклади: \7, \20 \377 (символ '\377' збігається з символом '0x00FF')

4 ТИПИ ДАНИХ

Всі типи даних Java діляться на дві категорії - примітивні і посилальні. Примітивні типи:

- числові типи
  - цілі
    - зі знаком byte, int, short, long;
    - без знака char;
  - речові float, double;
  - логічний тип boolean.
- посилальні типи:
  - класи (в т.ч. enum);
  - інтерфейси (в т.ч. анотації);
  - масиви;
  - Null-тип (представлений літералом null).

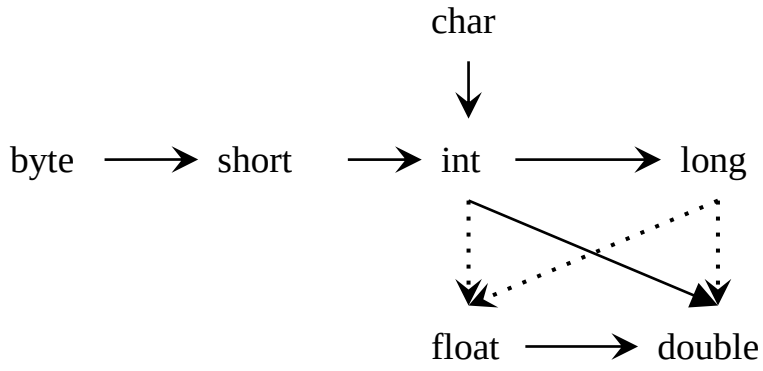
4.1 Примітивні типи даних

- Цілі числа
  - byte
    - логічна потужність: 1 байт
    - діапазон значень: [-128, 127]
    - логічна структура в бітах: перший біт вказує знак
    - приклади констант: відсутні
  - short
    - логічна потужність: 2 байта
    - діапазон значень: [-215, 215-1]
    - логічна структура в бітах: перший біт вказує знак
    - приклади констант: відсутні
  - int
    - логічна потужність: 4 байта
    - діапазон значень: [-231, 231-1]
    - логічна структура в бітах: перший біт вказує знак
    - приклади констант: 34 0x34 077 0b101
  - long
    - логічна потужність: 8 байт
    - діапазон значень: [-263, 263-1]
    - логічна структура в бітах: перший біт вказує знак
    - приклади констант: 34L 0x34l 077L 0b101L
  - char
    - логічна потужність: 2 байта
    - діапазон значень: [0, 232-1]
    - приклади констант: 'T' '\u000b' '\n' '\123'
- речові числа
  - float

        - логічна потужність: 4 байт
        - діапазон значень:
            - *приблизно [-3438, 3438]*
            - *infinity (нескінченність)*
            - *-infinity (мінус нескінченність)*
            - *NaN - не числиться*
        - логічна структура в бітах:
            - *1 біт вказує знак*
            - *8 біт вказують порядок*
            - *23 біта вказують мантиссу*
        - приклади констант: 34.4f 0.F 0x1.2p2f .7F
    - double
        - логічна потужність: 8 байт
        - діапазон значень:
            - *приблизно [-34308, 34308]*
            - *infinity (нескінченність)*
            - *-infinity (мінус нескінченність)*
            - *NaN - не числиться*
        - логічна структура в бітах:
            - *1 біт вказує знак*
            - *11 біт вказують порядок*
            - *52 біта вказують мантиссу*
    - приклади констант: 34.4 0. 0x1.2p2 .7
- логічний тип
    - boolean
        - логічна потужність: 1 біт
        - діапазон значень: {true, false}
        - приклади констант: true, false

4.2 Перетворення між примітивними типами даних

Між примітивними типами дозволені такі перетворення, які виконуються автоматично:



Пунктирними лініями відзначені дозволені автоматичні перетворення, при здійсненні які потенційно можлива втрата інформації. При перетвореннях, зазначених суцільними стрілками втрати інформації не відбувається.

Зауваження. Тип int може містити максимальну за величиною число, яке може бути записано в двійковій формі за допомогою 31-го біта. Тип float для запису цифр використовує мантиссу ємністю в 23 біта, а double - 52 біта. Тому, будь-яке число типу int поміститься в мантиссу змінної типу double (перетворення без втрат інформації), але існують такі числа типу int, які не вмістяться в мантиссу змінної типу float (перетворення з втратою інформації). Аналогічно, числа типу long, для запису яких застосовуються 64 біта, можуть не помістится в мантиссу речових змінних float (23 біта) і double (52 біта).

4.3 Явна приведення примітивних типів

Оператор приведення типів дозволяє здійснювати перетворення між будь-яким примітивними типами за винятком boolean. приклади:

- char ch = (char) -34L;
- byte b = (byte) 1234567L;
- byte b = (byte) 12345.6789

Такого роду перетворення здійснюються простим відкиданням "зайвих байт" і в більшості випадків будуть приводити до втрати інформації.

4.4 Тип результату арифметичного виразу

Результат арифметичного виразу, що містить змінні і константи, має тип, який визначається виходячи з типів входять до цей вислів компонент (змінних і констант).

Зауваження. Тип результату инкремента або декремента збігається з типом операнда.

Типи даних змінних і констант, що входять у вираз	Тип результату	приклад
byte, short, int, char в будь-якому поєднанні	int	'A' + 4
+ Long до попереднього в будь-якому поєднанні	long	'A' * 4L
+ Float до попереднього в будь-якому поєднанні	float	'A' * 4L / 34F
+ Double до попереднього в будь-якому поєднанні	double	34.4 - 'a'

4.5 Присвоєння константних виразів

Змінної типу byte, short або char можна привласнити значення константного виразу (виразу, в яке входять виключно тільки константи), якщо воно:

• має тип byte, short, char або int;
• не виходить за межі області значень відповідного типу.
приклади:

вираз	перетворення
`byte b = -54;`	`byte ← int`
`final short s = -4;`	`short ← int`
`b = s;`	`byte ← short`
`char ch = b;`	`char ← byte`
`ch = s;`	`char ← short`
`char ch = b + s;`	`char ← int`

Результат таких виразів визначає компілятор на етапі компіляції.

5 ОПЕРАЦІЇ І ОПЕРАТОРИ

5.1 Оператор присвоювання

Присвоєння змінній значення константи, іншої змінної або виразу, називається оператором присвоювання і позначається знаком "=", наприклад:

```
x = 7; y = x;
```

Припустимо багаторазове використання операції присвоювання в одному вираженні, наприклад:

```
x = y = z = 7;
```

Ця операція виконується справа наліво, тобто спочатку змінної z буде присвоєно значення 7, потім змінної y буде присвоєно значення змінної z, далі змінної x буде присвоєно значення змінної y.

5.2 Складовою оператор присвоювання

Загальний вигляд оператора

```
X op = Y;
```

Такий запис еквівалентна наступній:

```
X = (type of X) (X op Y);
```

де

- X - змінна;
- op - знак операції (один з: * /% + - << >> >>> & ^ |);
- Y - вираз.

Приклад операції присвоювання з операцією:

```
int x = 2;
x + = 2.5;
```

5.3 Арифметичні операції

	Назва	приклад
+	унарний плюс	byte x = 7; int y = + x;
-	унарний мінус	int x = -2;
++	інкремент	byte x = 1; x ++; ++ x;
-	декремент	byte x = 1; x--; --x;
+	додавання	int x = 2 + 3;
-	віднімання	int x = 7 - 5;
*	множення	long x = 4 * 5L;
/	розподіл	float x = 4F / 5.5f;

%	Залишок від ділення	double x = 3.4% 1.2

5.4 Операції порівняння

	Назва	Приклад
==	Так само	boolean b = 2 == 2;
! =	Не дорівнює	boolean b = 2! = 3;
>	більше	boolean b = 2> 3;
<	менше	boolean b = 2 <3;
> =	Більше або дорівнює	boolean b = 2> = 3;
<=	Менше або дорівнює	boolean b = 2 <= 3;

5.5 Логічні операції

	Назва	приклад
&	"І"	boolean b = true & false;
&&	"І" по короткій схемі	boolean b = true && false;
|	"АБО"	boolean b = true | false;
||	"АБО" по короткій схемі	boolean b = true || false;
^	Що виключає "АБО"	boolean b = true ^ false
!	заперечення	boolean b =! true;

5.6 Побітові операції

	Назва	Приклад
<<	зрушення вліво	int x = 2 << 3;
>>	Зрушення вправо зі знаком	int x = -4 >> 1;
>>>	Зрушення вправо без знака	int x = -4 >>> 1;
|	Побітовое "АБО"	int x = 2 | 3;
&	Побітовое "І"	int x = 5 & 0xFFFF;
^	Побітове виключає "АБО"	int x = 0 ^ -1;
~	побітове доповнення	int x = ~ -1;

5.7 Операція з'єднання з рядком +

Якщо одним з операндів операції '+' є рядок (об'єкт типу String), то другий операнд завжди буде приведений до рядка і результат буде представляти із себе рядок:

Тип операнда	Перетворення в рядок
Примітивний тип	Виклик статичного методу String.valueOf (x)
Вказівний тип	Виклик методу toString()
Літерал null	Буде підставлений строковий літерал "null"

5.8 Оператор порівняння типів instanceof

Загальний вигляд оператора порівняння типів:

```
X instanceof Type
```

де

- X - об'єкт;
- Type - тип.

Оператор повертає значення true, якщо X може бути автоматично приведений до типу Type і false в іншому випадку.

5.9 Тернарний оператор вибору

Загальний вигляд оператора:

```
B? X: Y
```

де B - логічне вираження.

оператор повертає

- X, якщо значення B одно true
- Y, якщо значення B одно false.

Даний оператор може служити заміною умовного оператору if / else.

5.10 Оператор приведення типу

Загальний вигляд оператора:

```
(Type) O
```

де Type - тип, до якого буде наведено операнд O.

Приклади приведення типів:

```
byte x = (byte) 34L;
char ch = (char) 34.34;
```

5.11 Оператор багатозначного вибору switch

Оператор switch передає управління одному або декільком секціях case в залежності від значення селектора. Приклад оператора:

```
switch (x) {
```

```
case 3:
case 4:
     System.out.println("3 or 4");
     break;
case 7:
     System.out.println("7");
     break;
default:
     System.out.println("Not 3, 4 or 7");
}
```

Тип селектора:

- enum
- String
- int і всі типи, які до нього приводили (byte, short, char).

У разі, якщо селектор має тип enum, в секціях case тип перечислимого типу не пишуть:

```
enum E {A, B, C}
...
E e = EA;
switch (e) {
case A:
case B:
     System.out.println("A or B");
     break;
default:
     System.out.println("Not A nor B");
}
```

6 КЛАСИ

Клас - це шаблон, екземпляр класу - реалізація шаблону.

6.1 Вміст класу

Клас може містити:
- конструктори
- блоки ініціалізації
- методи
- поля
- вкладені типи

6.2 Види класів по оголошенню

Всього існує два види класів по оголошенню: звичайний class і так званий. перелічувальний тип enum.

6.3 Види класів по розташуванню

Залежно від місця оголошення класу розрізняють класи:
- верхнього рівня;
- вкладені:
  - внутрішні;
  - елементи класів;
  - локальні;
  - анонімні.

6.4 Просте і повне ім'я класу

Кожен клас має коротке ім'я і повне ім'я (так зване повне кваліфіковане ім'я - Full Qualified Name, FQN). Остання являє собою коротке ім'я попереджання ім'ям того пакету, в якому розташований клас. Наприклад, після такого оголошення:

```
package com.my;
class A {...}
```

клас A має просте ім'я 'A' і повне ім'я 'com.my.A'.

6.5 Пакети

Пакети визначають простору імен типів. Пакети можуть бути вкладеними. Для того, щоб імпортувати типи з деякого пакету, досить написати декларацію import:

```
// імпорт одного типу A
import com.my.A;

// імпорт типу Entry
// який вкладений в тип java.util.Map
import java.util.Map.Entry;
```

```
// імпорт всіх типів з пакета java.util
import java.util. *;

// імпорт всіх статичних полів і методів
// класу java.util.Arrays
import static java.util.Arrays. *;
```

Приклади пакетів з ядра Java:

Назва пакета	вміст
java.lang	базові типи
java.util	структури даних
java.io	потоки введення / виводу
java.sql	JDBC
javax.swing	GUI

6.6 Статичні елементи класу

Статичні елементи класу належать класу, але не його екземплярів. Приклад статичних сутностей:

```
class A {
     static int x; // статичне поле
     static void m() {...} // статичний метод
     static {...} // // статичний блок ініціалізації
     static class B {...} // статичний вкладений тип
}
```

З статичного контексту не можна звертатися до не статичною. Наступний код не відкомпілюйте:

```
class A {
     int x;
     static void m() {int y = x;}
}
```

6.7 Конструктори класу

конструктори:

- створюють екземпляр класу;
- мають ім'я збігається з простим ім'ям класу;
- не можуть бути успадковані;
- не мають тип повертається результату;
- можуть мати будь-який рівень доступу.

Приклад використання конструктора:

```
class Test {
```

```
    Test() {...}
}
Test t = new Test();
```

Оператор new є оператором створення екземпляра класу.

6.8 Конструктор за замовчуванням

Конструктор за замовчуванням - це конструктор без параметрів:

```
public class A {
    public A() {...}
}
```

Якщо в класі не визначено жодного конструктор, то компілятор створить і вставить в байт код конструктор за замовчуванням (такий конструктор містить виклик конструктора за замовчуванням класу предка). Таким чином, будь-який клас містить принаймні один конструктор.

Рівень доступу конструктора, який автоматично генерує компілятор збігається з рівнем доступу класу.

6.9 Методи класу

Методи визначають функціональність. Можуть бути статичними або не статичною. Приклад виклику методу:

```
class Test {
     void m() {...}
}
...
Test t = new Test();
t.m();
```

6.10 Перевантаження методів

Перевантажити метод (точніше, по специфікації, - перевантажити ім'я методу) означає оголосити ще один метод з тим же самим ім'ям, але іншим списком параметрів (враховуючи порядок проходження параметрів).

Приклад:

```
class A {
     void m() {...}
     void m(int x) {...}
}
```

Зауваження. Конструктори класу (якщо їх більше одного) завжди перевантажені.

6.11 Поля класу

Значення полів класу визначають стан об'єкта. Приклад використання полів:

```
class Human {
     int age = 30;
}
...
Human human = new Human();
System.out.println (human.age);
```

6.12 Ініціалізація полів класу

Поля в класі можуть бути проініціалізовані:

- при оголошенні;
- в конструкторах;
- в блоках ініціалізації;
- в методах

6.13 Значення полів класу за замовчуванням

Поля класу, що не були проініціалізовані, містять значення за замовчуванням:

Тип	Значення
byte, short, int, long, char, float, double	0
boolean	false
Нормативний	null

6.14 Блоки ініціалізації

Блоки ініціалізації ініціалізують об'єкт.

```
class Test {
     {...}
}
```

або клас

```
class Test {
     static {...}
}
```

Блоків ініціалізації може бути кілька.

Вміст нестатичних блоків ініціалізації буде виконано в порядку їх появи в тексті програми разом з ініціалізатор нестатичних полів.

Вміст статичних блоків ініціалізації буде виконано в порядку їх появи в тексті програми разом з ініціалізатор статичних полів.

Статичні блоки ініціалізації виконує JVM при першому завантаженні класу. Нестатичні - при створенні нового об'єкта перед тим, як буде виконано тіло конструктора.

6.15 Ключове слово this

Ключове слово this може бути використано в кількох контекстах:

• це посилання на екземпляр класу, який її використовує (об'єкт this):

```
class A {
    A() {...}
    A(int x) {
        this();
    }
}
```

• з його допомогою можна викликати один конструктор з іншого (такий виклик, якщо він є, завжди повинен бути першим рядком конструктора):

```
class A {
    A() {...}
    A(int x) {
        this();
        // ...;
    }
}
```

• з його допомогою можна звернутися з внутрішнього класу до об'єкту this зовнішнього класу:

```
class A {
     class B {
          void m() {
               // an object of the outer class
               System.out.println(A.this);

               // an object of the inner class
               System.out.println(this);
          }
     }
}
```

6.16 Ключове слово super

• З його допомогою можна звернеться до елементу класу предка:

```
class A {
    int x = 7;
}
class B extends A {
    int x;
    void m() {
        x = x + super.x;
    }
}
```

• з його допомогою можна викликати конструктор класу предка (такий виклик, якщо він є повинен бути першим рядком конструктора):

```
class A {
    A (int x) {...}
}
```

```
class B extends A {
    B() {
        super (5);
    }
}
```

6.17 Виклик конструктора предка з конструктор нащадка

Будь-конструктор завжди містить першим рядком виклик конструктор предка. Якщо виклик явно не написаний, то компілятор вставить в байт код виклик

```
super();
```

7 ООП В JAVA

7.1 Спадкування

Спадкування - це відношення між посилальними типами (класами або інтерфейсами)), яке задовольняє критерію "окремий випадок". Тип нащадка завжди є окремим випадком типу предка. Нащадок може замінити предка в будь-якому контексті.

Приклад успадкування:

```
class A {...}
class B extends A {...}
```

7.2 Інкапсуляція

Інкапсуляція полягає в:

- обмеження доступу до елементів класу;
- приховуванні деталей внутрішньої реалізації.

Приклад інкапсуляції:

```
class Human {
    private int age;
    public void setAge (int age) {
        if (age> 0 && age <120) {
            this.age = age;
        }
    }
}
```

Мета інкапсуляції - цілісність об'єкта.

7.3 Поліморфізм

Поліморфізм це властивість платформи, яке полягає в тому, що змінної має тип предка можна привласнити посилання на об'єкт - екземпляр класу нащадка, прим цьому потім може мати свою власну реалізацію функціональності.

Приклад поліморфізму:

```
class Base {
    void m() {...}
}
class A extends Base {
    void m() {...}
}
class B extends Base {
    void m() {...}
}

Base b1 = new A(); b1.m();
Base b2 = new B(); b2.m();
```

Зауваження. Поля, статичні методи, а також фіналізовані методи поліморфними не є, для них поліморфізм "не працює".

7.4 Рівні доступу до елементів класу

Специфікатор	Назва	Область видимості елемента
public	публічний	З будь-якого зовнішній коду
protected	захищений	З того ж пакету і зсередини будь-яких нащадків
Відсутнє	за замовчуванням	З того ж пакета
private	приватний	Усередині поточного класу

7.5 Рівні доступу до класів

Можливі рівні доступу, які можна надавати класах залежать від виду класу:

Виду класу	Рівні доступу
Верхнього рівня	public, default
Внутрішні (за винятком локальних)	public, default, protected, private
Локальні	default *

* Для локальних класів рівень доступу default (тобто, коли специфікатор рівня доступу не присутній взагалі) означає видимість тільки в межах того блоку коду, в якому визначено клас.

7.6 Перекриття методів і сигнатура

Сигнатурою методу називають ім'я методу і упорядкований список типів його аргументів. Перекрити метод означає оголосити в типі нащадку метод з

тієї ж самої сигнатурою, що і тип предок. Перекриття методів дозволяє реалізувати поліморфізм.

```
class A {
    void m() {...}
}
class B extends A {
    void m() {...}
}
```

Зауваження. При перекритті методу предка на метод нащадка накладені такі обмеження:

- не можна звужувати рівень доступу;
- тип повертається результату повинен бути узгоджений з таким методу предка:
  - для примітивних типів і void - повинен збігатися;
  - для посилальних - бути автоматично приводиться;
- не можна розширювати безліч перевірених викидаються винятків.

7.7 Приховування методів

Для статичних методів відсутнє таке поняття як перекриття методів (вони не поліморфні). Якщо оголосити в типі нащадку метод з такою ж сигнатурою як і у методу типу предка, то таку ситуацію називають приховуванням (hide) методу.

```
class A {
    static void m() {...}
}
class B extends A {
    static void m() {...}
}
```

Статичні методи належать типу, а не екземплярам типу. Якщо виклик статичного методу буде здійснений на об'єкті, то такий запис просто буде замінена викликом методу на типі.

Приклад:

```
class A {
    static void m() {
        System.out.println("A # m");}
    }
...
A a = null;
a.m(); // буде виведено A # m
a.m(); //// буде виведено A # m
```

При приховуванні методів діють ті ж обмеження, що і при перекритті методів. Також не можна приховати не статичною метод або ж перекрити статичний метод.

7.8 Ключове слово final

Ключове слово final може бути використано в наступних контекстах:

Сутність	Властивості які набуває сутність
Клас	Не можна успадковувати
Метод	Не можна перекривати
Статичний метод	Не можна приховувати
Поле	Є константою
Локальна змінна	Можна форматувати максимум один раз
Параметр методу	Не можна змінити значення

7.9 Фіналізовані поля

Поле оголошене з специфікатором final є константною. Такі поля повинні бути в обов'язковому порядку проініціалізірованни.

Поле	Місця, де можна ініціалізувати
Чи не статичне	Пи оголошенні В конструкторі В блоці ініціалізації
статична	При оголошенні В статичному блоці ініціалізації

7.10 Порядок виклику блоків ініціалізації і конструкторів

Статичні і не статичні блоки ініціалізації а також вміст конструкторів класів при наявності ієрархії успадкування при створенні об'єкта будуть викликані в строго визначеному порядку.

Приклад.

Нехай є ієрархія (напрямок стрілки від класу нащадка до класу предка):

```
Z ⇒ ... ⇒ B ⇒ A
```

Тоді JVM при створенні об'єкта Z здійснить наступне.

1. Послідовно перебере всі класи A, B, ..., Z і в кожному з них виконає в порядку їх появи в класі всі:
   - статичні ініціалізатор полів,
   - статичні блоки ініціалізації;

за винятком тих, які JVM вже раніше виконувала.

2. Послідовно перебере всі класи A, B, ..., Z і в кожному з них виконає:

1) в порядку їх появи в класі:
   - ініціалізатор полів,
   - блоки ініціалізації;

2) тіло конструктора.

Приклад:

```
class A {
     static {System.out.println("static block A"); }
     {System.out.println("block blockA"); }
     A() {System.out.println("constr A"); }
}

class B extends A {
     static {System.out.println("static block B"); }
     {System.out.println("block blockB"); }
     B() {System.out.println("constr B"); }
}

class Z extends B {
     static {System.out.println("static block Z"); }
     {System.out.println("block block Z"); }
     Z() {System.out.println("constr Z"); }
}
```

Якщо створити екземпляр класу Z, то висновок буде таким:

```
new Z();

результат:
static block A
static block B
static block Z
block blockA
constr A
block blockB
constr B
block block Z
constr Z
```

Зауваження. Вся статика буде виконана в момент першого завантаження класу в систему.

```
new B();
System.out.println("~~~");
new Z();

результат:
static block A
static block B
block blockA
constr A
block blockB
constr B
~~~
static block Z
block blockA
constr A
block blockB
```

```
constr B
block block Z
constr Z
```

7.11 Перетворення типів між класами

Між класами, які перебувають у відношенні спадкування можливо перетворення типів двох видів: висхідний і спадний:

- висхідний перетворення типів це перетворення від нащадка до предка, воно може виконуватися неявно;
- спадний перетворення типів може здійснюватися від класу предка до класу, який знаходиться в будь-якому вузлі ієрархії успадкування від пращура до нащадка.

8 ОБОЛОНКИ І ВКЛАДЕНІ КЛАСИ

Вкладені класи - це класи, визначення яких знаходиться всередині оголошень інших типів.

8.1 Класифікація вкладених класів

Вкладені класи можуть бути представлені у вигляді наступних категорій:

- класи - елементи класу
  - статичні
  - не статичною
- локальні класи
  - із зазначенням імені
  - анонімні

Приклад:

```
class A {// клас верхнього рівня
    class B {} // клас елемент класу
    void m() {
        class C {} // локальний клас
        m (new SomeClass() {  // анонімний клас
            ...
        });
    }
}
```

Вкладені класи, які не є статичними називають внутрішніми.

8.2 Класи - елементи класів

Класи, оголошення яких знаходиться на рівні оголошень елементів класу (полів, методів) можуть мати такі модифікатори і специфікатор:

- abstract
- static
- final
- private, protected, public

8.3 Локальні класи

Локальними класами є класи, оголошення яких знаходиться всередині методів, конструкторів, блоків ініціалізації.

Рівень доступу локальних класів - за замовчуванням, тобто специфікатор доступу не ставлять, але для них цей рівень доступу означає видимість в межах блоку коду, в якому вони визначені (найближчі обрамляють фігурні дужки).

8.4 Анонімні класи

Анонімний клас - локальний клас, який не має імені. Анонімний клас завжди розширює клас або реалізує інтерфейс. Часто анонімні класи

використовують для створення "на льоту" примірника реалізації інтерфейсу або спадкоємця класу.

приклади:

```
abstract class A {
    abstract void m();
}
...
// a - екземпляр анонімного класу,
// який успадковує клас A
A a = new A() {
    void m() {...}
};
```

При створенні об'єкта за допомогою анонімного класу можна викликати конструктор з параметром:

```
class A {
    A (int x) {}
}
...
A a = new A (7) {
    ...
};
```

8.5 Властивості внутрішніх класів

Деякі властивості внутрішніх класів:

- не можуть оголошувати статичних полів (крім констант), методів і класів (але можуть наслідувати їх;
- мають доступ до всіх елементів зовнішнього класу (в т.ч. приватним);
- мають доступ до локальних змінних і параметрів методу (вони повинні бути оголошені як final).

8.6 Створення об'єктів вкладених класів

Для створення екземплярів внутрішніх класів використовують розширений синтаксис оператора new:

```
class A {
    class B {}
}
...
A a = new A();
A.B b = a.new B();
```

Створення екземплярів вкладених статичних класів не відрізняється від створення екземплярів класів верхнього рівня, тільки слід врахувати, що їх ім'я включає ім'я зовнішнього класу:

```
class A {
    static class B {}
}
```

```
...
A.B b = new A.B();
```

8.7 Доступ до об'єкту зовнішнього класу з внутрішнього

Для доступу до об'єкта зовнішнього класу з внутрішнього використовують ключове слово this:

```
class A {
    private int x;
    class B {
        int x = A.this.x;
    }
}
```

8.8 Класи оболонки

Кожен примітивний тип має відповідний йому клас оболонку.

Примітивний тип	Клас оболонка
byte	Byte
short	Short
int	Integer
long	Long
char	Character
float	Float
double	Double
boolean	Boolean

Специфікаторами void, який використовують для того, щоб показати, що метод не повертає значення (так званий "псевдотіпов" void), поставлений у відповідність клас оболонка Void.

Boolean, Character, Void успадковують Object безпосередньо, інші класи оболонки успадковують клас Number, який є спадкоємцем класу Object.

8.9 Перетворення між примітивами і їх оболонками

Якщо в операції повинен брати участь об'єкт, а бере участь примітив, то цей примітив автоматично буде обгорнутий в об'єктну оболонку (так званий autoboxing):

```
// еквівалентно Integer x = Integer.valueOf (7)
Integer x = 7;

// еквівалентно Boolean b = Boolean.valueOf (true);
Boolean b = true;
```

Якщо в операції повинен брати участь примітив, то замість нього можна підставити об'єктну оболонку. Значення буде автоматично з неї розгорнуто (так званий unboxing):

```
// еквівалентно: int y = x.intValue() + 2;
Integer x = new Integer (7);
int y = x + 2;

// еквівалентно
// boolean f =! B.booleanValue() ^ false;
Boolean b = new Boolean (true);
boolean f =! b ^ false;
```

9 АБСТРАКТНІ КЛАСИ ТА ІНТЕРФЕЙСИ

9.1 Абстрактні методи

Абстрактний метод - це метод не має реалізації. Такі методи повинні бути оголошені з модифікатором abstract.

```
abstract void m();
```

Зауваження. Для методів не допускається поєднання abstract з модифікаторами final, private і static.

9.2 Абстрактні класи

Абстрактний клас - це клас оголошений зі специфікатором abstract.
Якщо клас містить хоча б один абстрактний метод, він також повинен бути абстрактним. Однак абстрактний клас не зобов'язаний утримувати абстрактні методи.

Приклади абстрактних класів:

```
abstract class A {
    abstract void m();
}

abstract class B {}
```

Зауваження. Для класів не допускається поєднання abstract з модифікатором final.

Призначення абстрактних класів:

- інтерфейс до сімейства класів;
- база для реалізації поліморфізму.

Як правило, в абстрактних класах визначають частково функціональність, залишивши частину методів не реалізованими для спрощення подальшої реалізації; при спадкуванні, клас нащадок наповнює функціональністю тільки нереалізовані методи.

Властивості абстрактних класів:

- не можна інстанціювать, але можна оголосити змінну даного типу;
- може мати конструктори;
- може мати не абстрактні методи;
- може не містити абстрактних методів.

9.3 Спадкування абстрактного класу

Клас, який успадковує абстрактний клас повинен або реалізувати всі його абстрактні методи або, в іншому випадку, повинен бути оголошений абстрактним.

```
abstract class A {
    abstract void m();
}
```

```
// помилка компіляції:
class B extends A {}
```

9.4 Об'єктні змінні абстрактних класів

Не можна створити екземпляр абстрактного класу, однак можна оголосити об'єктну змінну цього класу і змусити посилатися її на об'єкт - екземпляр класу, який успадковує і реалізує даний абстрактний клас.

```
abstract class A {
    abstract void m();
}

public class B extends A {
    void m() {
        System.out.println (m());
    }
}
```

9.5 Інтерфейси

Інтерфейси визначають:

- кордони взаємодії між об'єктами;
- абстракцію, реалізацію якої надає імплементує інтерфейс сторона.

Приклад інтерфейсу:

```
interface G {
    int x = 7;
    void m();
}
```

Використовувати інтерфейс за допомогою таких дій:

- клас може реалізовувати інтерфейс;
- можна оголосити интерфейсную змінну;
- інтерфейс може успадковувати кілька інших інтерфейсів.

На відміну від класів, де множинне спадкування класів заборонено, для інтерфейсом множинне спадкування дозволено.

```
interface G extends G2, G3 {
    ...
}
```

9.6 Елементи інтерфейсу

Інтерфейси можуть містити:

- поля (public static final);
- методи (public abstract);
- інтерфейси (public static);
- класи (public static).

Зазначені модифікатори і специфікатор можна не ставити, вони маються на увазі за замовчуванням.

Поля інтерфейсу є:

- константами (final);
- статичними (static);
- публічними (public);
- повинні бути проініціалізовані при оголошенні.

Методи інтерфейсу є:

- абстрактними (abstract);
- публічними (public).

Вкладені класи і інтерфейси є статичними.

9.7 Реалізація інтерфейсу

Клас може реалізовувати повністю або частково один або кілька інтерфейсів:

```
class A implements G, G2 {...}
```

Клас може одночасно успадковувати інший клас і реалізовувати інтерфейси:

```
class A extends B implements G, G2 {...}
```

Клас може частково реалізовувати інтерфейс, в такому випадку він повинен бути оголошений абстрактним:

```
interface G {
     void m();
     void m2();
}

abstract class A implements G {
     // метод повинен бути публічним!
     public void m() {}
}
```

За допомогою оператора instanceof можна перевірити, чи є об'єкт екземпляром класу, що реалізовує інтерфейс:

```
interface G {}
class A implements G {}
...
System.out.println (new A() instanceof G); // true
```

10 СТРОКОВІ КЛАСИ

10.1 Клас String

Клас String є незмінну рядок: створений об'єкт цього класу не може бути змінений. Даний клас не можна успадковувати, тому що він оголошений зі специфікатором final.

10.2 Змінні рядка

До змінним рядках відносять принаймні два класи StringBuilder і StringBuffer. Вони практично ідентичні за своїм змістом, відміну же полягає в тому, що StringBuffer містить синхронізовані методи на відміну від StringBuilder, де методи не синхронізація.

При необхідності конкатеніровать рядки в циклі доцільно використовувати змінні рядки замість об'єктів типу String:

```
String str = "asdf";
StringBuilder sb = new StringBuilder();
for (int j = 0; j <100_000; j ++) {
    sb.append (str);
}
```

StringBuilder і StringBuffer як і клас String є фіналізували, тобто їх не можна успадковувати. Також вони реалізують шаблон проектування Builder:

```
StringBuilder sb = new StringBuilder()
    .append("asdf")
    .append('c')
    .append (34);
```

10.3 Інтерфейс CharacterSequence

String, StringBuilder, StringBuffer реалізують інтерфейс CharacterSequence, який містить загальні для цих класів методи:

```
// повертає символ в позиції index
// нумерація символів рядка йде з 0
public char charAt (int index)

// повертає довжину рядка
public length()

// повертає підрядок
// від start включно до end виключно
subSequence (int start, int end)
```

10.4 Рівність рядків

Для визначення равнества рядків посимвольний можна використовувати метод equals класу String (в StringBuilder / Buffer даний метод не перекритий, а

"дістається в спадок" від класу Object, який порівнює об'єкти за допомогою операції "=="):

```
public boolean equals (Object object)
```

Метод equals повертає значення true, якщо рядок, на якій викликаний метод містить в точності таку ж послідовність символів, як і рядок, яка передана методу в якості аргументу, в іншому випадку метод поверне значення false.

Для порівняння рядків без урахування регістру символів, з яких складається рядок, можна використовувати метод equalsIgnoreCase класу String.

```
public boolean equalsIgnoreCase (String anotherStr)
```

10.5 Порівняння рядків

Для порівняння рядків можна використовувати метод compareTo класу String, який лексикографічно порівнює поточну рядок з рядком аргументом:

```
public int compareTo (String anotherStr)
```

10.6 Пул рядків

Екземпляри класу String є незмінними об'єктами. Можна створити безліч строкових об'єктів, які містять ідентичний набір символів, але не рівні один одному з точки зору операції "==" (тобто, займають різні ділянки в пам'яті).

JVM підтримує пул (контейнер) рядків, який містить унікальні за своїм символьному набору рядки. Якщо деяка рядок міститься в пулі, то її називають канонічним поданням всіх рядків, які мають той же символьний склад.

Метод intern класу String призначений для пулірованія (занесення в пул) рядків. Якщо на рядку викликати метод intern, то він перегляне пул рядків і, використовуючи метод equals, спробує знайти рядок у пулі, еквівалентну даній. Якщо такий рядок буде знайдена, то intern поверне на неї посилання, в іншому випадку метод занесе в пул посилання на вихідну рядок і поверне її як результат виклику.

Якщо змінної типу String привласнити результат виклику на ній же методу intern:

```
str = str.intern();
```

то ця змінна гарантовано буде посилатися на пулірованную рядок

Пулірованние рядки можна порівнювати оператором "==", що може прискорити порівняння рядків. З іншого боку пуліровать взагалі все рядки недоцільно, тому що велика кількість рядків в пулі уповільнює перегляд пулу методом intern при пулірованіі рядки.

Зауваження. Метод String # equals на самому початку містить код

```
if (this == anObject) {
    return true;
}
```

тому для пулірованних рядків замінювати порівняння за методом equals порівнянням за допомогою операції "==" сенсу немає (останнє вже міститься в першому).

10.7 Конкатенация строк

В Java відсутня перевантаження операцій в сенсі перевизначення дії операцій, але існує в сенсі застосування одного знака операції до різних типів даних.

Якщо в операції додавання бере участь рядок, то інший операнд завжди буде приведений до рядка. При конкатенації рядків буде створений новий об'єкт типу String, якщо хоча б один аргумент операції "+" не є константою, в іншому випадку з'єднання рядків буде здійснено вже на етапі компіляції.

```
final String s = "asdf";
String s2 = "7";

// true
System.out.println (s + 7 == "asdf7");

// false
System.out.println (s + s2 == "asdf7");
```

10.8 Метод Object # toString

Кожен клас має успадкований від супертіпа Object публічний метод toString:

```
public String toString()
```

який повертає строкове представлення об'єкту.

У реалізації JSE від компанії Oracle метод класу Object повертає повне ім'я класу, знак "@" і шістнадцяткове представлення хеш-коду об'єкта.

11 РЕГУЛЯРНІ ВИРАЗИ

Регулярні вирази - формальна мова пошуку і здійснення маніпуляцій з підрядками в тексті. Заснований на використанні метасимволов.

11.1 Символи

Метасимвол	Значення
x	символ x
\\	зворотний слеш
\xhh	символ з кодом U + 00hh
\xhhh	переклад рядка
\n	переклад рядка
\r	повернення каретки
\t	табуляція

11.2 Символьні класи

Конструкція	Значення	Приклад		
		Вхід	Рег. вир.	Що знайдено
[abc]	a, b або c	accddba	[ab]	**a** ccdd **b** **a**
[^abc]	будь-який сімволкроме a, b, c	acdba	[^ab]	a **c** **d** ba
[a-zA-Z]	будь-який символ від a до z або від A до Z	adAcdh	[a-cA-Z]	**a** d **A** **c** dh
[a-z[A-Z]]	то ж, що і [a-zA-Z]	-		
[a-z&&[def]]	d, e або f	accdddeab	[a-d&&c-f]	a **c** **c** **d** **d** **d** eab
[a-z&&[^def]]	будь-який символ від a до c або від g до z	accdddeab	[a-d&&[^c-f]]	**a** ccddde **a** **b**

11.3 Символьні класи Java

Символьні класи Java визначають символи, для яких відповідні статичні методи класу оболонки Character повертають значення true:

Символьний клас Java	Метод класу Character
\p{javaLowerCase}	isLowerCase
\p{javaUpperCase}	isUpperCase
\p{javaWhitespace}	isWhitespace

Приклад:

Вхід	Текст
Регулярний вираз	\p{javaUpperCase}\p{javaLowerCase}
Знайдені відповідності	**Те** кст

11.4 Зумовлені класи

Конструкція	Значення	Приклад		
		Вхід	Рег. вир.	Що знайдено
.	будь-який символ	abcdef gh	...	**abc def** gh
		abcde	..	**ab cd** e
\d	цифра (тобто [0-9])	ab8ab8	\d\D	ab **8a** b8
\D	цифра (тобто [^ \d])			
\s	символ пробілу еквівалент: [\t\n\f\r\x0b]	ab 8 ab8	\s\S	ab **8 a** b8
\S	непробельний символ, [^ \s]			
\w	символ слова, [a-zA-Z_ \d]	ab*8&a b8	\w\W	a **b* 8&** ab8
\W	заперечення \w, [^ \w]			

11.5 Межі

Конструкція	Значення	Приклад		
		Вхід	Рег. вир.	Що знайдено
^	початок рядка	abababab	^ab	**ab** abab

$	кінець рядка	ababab	ab$	abab **ab**
\b	межа слова	abc abcd	abc\b	**abc** abcd
\B	заперечення \b	abc abcd	abc\B	abc **abc** d
\A	початок введення	abc abc abc abc	\Aabc	**abc** abc abc abc
\z	кінець введення		abc\z	abc abc abc **abc**
\Z	кінець введення, в т.ч. обмежувач рядка	abc abc abc abc<LF>	abc\Z	abc **abc<LF>**

11.6 Обмежувачі рядків

Обмежувач	Назва
'\n'	LF (новий рядок)
'\r'	CR (повернення каретки)
"\r\n"	CR + L
'\u0085'	наступний рядок
'\u2028'	роздільник рядка
'\u2029'	роздільник параграфа

11.7 Квантіфікатори

Квантіфікатори визначають повторюваність.

Жадібний квантіфікатор визначає максимально можливу подстроку. Лінивий квантіфікатор визначає мінімально можливу подстроку.

Конструкція	Значення	Вид	Приклад		
			Вхід	Рег. вир.	Що знайдено
X?	один або нуль раз	жадібний	aabcabbb	ab?	**a** **ab** c **ab** bb
X??		ледачий		ab??	**a** **a** bc **a** bbb

X*	нуль або більше разів	жадібний	aabcabbb	ab*	**a** **ab** c **abbb**
X*?		ледачий		ab*?	**a** **a** bc **a** bbb
X+	один або більше разів	жадібний	aabcabbb	ab+	a **ab** c **abbb**
X+?		ледачий		ab+?	a **ab** c **ab** bb
X{n}	рівно n раз	жадібний	aabcabbb	ab{2}	aabc **abb** b
X{n}?		ледачий		ab{2}?	
X{n,}	Проте n раз	жадібний	aabcabbb	ab{2,}	aabc **abbb**
X{n,}?		ледачий		ab{2,}?	aabc **abb** b
X{n,m}	від n до m раз	жадібний	aabcabbb	ab{1,2}	a **ab** c **abb**b
X{n,m}?		ледачий		ab{1,2}?	a **ab** c **ab** bb

11.8 Сверхжадние квантіфікатори

Крім жадібних і ледачих квантіфікаторов існують ще т.зв. сверхжадние (або ревниві) квантіфікатори. Використовують їх не часто, однак найважливішою їх перевагою є набагато більш висока швидкість роботи в порівнянні з жадібними квантіфікаторамі.

Для того, щоб зробити жодній квантіфікатор сверхжадним досить додати "+" праворуч від квантіфікатора:

Жадібний квантіфікатор	Сверхжадний квантіфікатор
X?	X?+
X*	X*+
X+	X++
X{n}	X{n}+
X{n,}	X{n,}+
X{n,m}	X{n,m}+

Приклад.

При пошуку в рядку aab за допомогою регулярного виразу a + b кроки аналізатора будуть наступними:

Частина, що рег. вираження	Проверяемая частина входу

a+	a
a+	aa
a+	aab
aab не задовольняє рег. висловом a +, тому b буде "повернуто" назад, до вживаного регулярному виразу буде додана наступна конструкція і з входу знову буде прочитаний символ b.	
a+b	aab
Останній символ введення (b) прочитаний, відповідність знайдено.	

Сверхжадний квантіфікатор працює як жадібний, але ніколи не "повертає" прочитані символи назад.

При пошуку в рядку aab за допомогою регулярного виразу a ++ b кроки аналізатора будуть наступними:

Частина, що рег. вираження	Проверяемая частина входу
a++	a
a++	aa
a++	aab
Останній символ введення (b) прочитаний, відповідність, не знайдено.	

11.9 Логічні операції

У регулярних виразах можна використовувати логічні операції диз'юнкції (логічне "І") і кон'юнкції (логічне "АБО"):

конструкція	значення
XY	X за яким слід Y (AND)
X|Y	X або Y (OR)

Приклад:

Вхід	aabcabbb
Регулярний вираз	aa | b
знайдені відповідності	**aa** b ca bbb

Зауваження. Пріоритет AND вище ніж OR.

11.10 Групи

Вираз в круглих дужка - група. Кожна група має номер.

Групи нумерують зліва направо, починаючи з одиниці, при цьому номер може бути більше 9.

Щоб група не була пронумерована, вона повинна починатися з (?:

```
(A) (B (C) (?: D))
```

(A)	група номер 1
(B (C) (?: D))	група номер 2
(C)	група номер 3
(?: D)	група без номера

Групи можуть бути використані за номером в регулярному виразі за допомогою синтаксису: \НОМЕР_ГРУППИ.

Приклад:

Вхід	aabaab
Регулярний вираз	(Aab)\W\1
знайдені відповідності	**aab** aab

11.11 Екранування символів

Для представлення символів, які вдають із себе елементи синтаксису мови регулярних виразів використовують екранування за допомогою зворотного слеша:

Символ	\	|	+	*	?	^	$	(	)	[	]	{	}
Екранування	\\	\\|	\+	*	\?	\^	\$	\(	\)	\[	\]	\{	\}

Зауваження. Використання екранування не завжди необхідно:

Вхід	asd	(a)s{d}
Регулярний вираз	[^^]	[(){}]
Знайдені відповідності	**a s d**	**(**a**)**s**{**d**}**

Для вказівки діапазону екранування можна використовувати \Q і / або \E

конструкція	значення
\Q	початок діапазону
\E	закінчення діапазону

Приклад:

Вхід	ab\(*aa
Регулярний вираз	\Q\(*\E(a)\1
Знайдені відповідності	ab**\(*aa**

11.12 Попереджуючий перегляд вперед, перегляд назад

	Конструкція	Вид	Приклад		
			Вхід	Регулярне вираз.	Що знайдено

Перегляд вперед (look ahead)	(?=X)	позитивний	abacab	a(?=b)	**a**bacab
	(?!X)	негативний		a(?!b)	ab**a**cab
Перегляд назад (look behind)	(?<=X)	позитивний		(?<=b)a	ab**a**cab
	(?<!X)	негативний		(?<!b) a	**a**bac**a**b

11.13 Режими

Режими впливають на роботу регулярних виразів. Кожен режим має буквений код.

Режим	Код	Опис
COMMENTS	x	Режим коментарів. Пробільні символи ігноруються, після символу # можна писати коментар до регулярному виразу.
CASE_INSENSITIVE	i	Ігнорує регістр символів (діапазон ASCII)
UNIX_LINES	d	Роздільник рядків тільки CR (\r).
DOTALL	s	Точка (.) Може включати \n
UNICODE_CASE	u	Розширює дію CASE_INSENSITIVE на весь діапазон Unicode.
MULTILINE	m	Складний режим (за замовчуванням $ - кінець введення).

Щоб включити режим, досить випередити регулярний вираз комбінацією:

```
(? КОД_РЕЖІМА)
```

Якщо потрібно включити відразу кілька режимів, то можна писати кілька опцій.

Приклад:

Вхід	Регулярні вирази	Що знайдено
Юю	ю	**ю** Ю
	(?iu)ю	**ю** **Ю**

12 ІНТЕРНАЦІОНАЛІЗАЦІЯ, ЛОКАЛІЗАЦІЯ

12.1 Інтернаціоналізація

В англійській мові для слова "internationalization" прийнято скорочення "i18n". При цьому число 18 означає кількість пропущених між «i» та «n» букв:

```
I18n = I(nternationalizatio)n
```

Інтернаціоналізація - підтримка виведення значень параметрів у вигляді, який прийнятий в тій чи іншій країні. Як приклади параметрів можуть виступати наступні:

- грошові одиниці;
- час, дати, часовий пояс;
- одиниці заходів;
- телефонні номери;
- заголовки;
- поштові адреси та індекси.

12.2 Локалізація

В англійській мові для слова "localization" прийнято скорочення "l10n"

```
L10n = L(ocalizatio)n
```

Локалізація - адаптація програми до мови і ринку країни, де воно буде продаватися або використовуватися. При цьому можуть бути здійснені наступні маніпуляції:

- переклад документації, меню, повідомлень, файлів онлайн допомоги;
- зміна кольорів призначеного для користувача інтерфейсу відповідно до культурними традиціями у відповідній країні;
- зміна алгоритмів алфавітного сортування;
- додавання або зміна залежать від регіонального ринку компонентів.

12.3 Різниця між i18n і i10n

i18n - адаптація продукту для потенційного використання практично в будь-якій країні, а L10n - адаптація продукту для використання в конкретній країні.

12.4 Клас Locale

Основними класами для роботи з локалізацією і інтернаціоналізацією є наступні:

```
java.util.Locale
java.util.ResourceBundle
```

Локаль (клас Locale) - це ідентифікатор регіону. Пакет ресурсів (клас ResourceBundle) - сукупність ресурсів специфічних для певної локалі.

12.5 Створення об'єкта локалі

Конструктор в загальному випадку приймає три параметри:

```
language - lowercase two-letter ISO-639 code.
country - uppercase two-letter ISO-3166 code.
variant - варіант мови
```

Всього існує три конструктора:

Локаль заснована на	конструктор
мовою	`Locale(String language)`
Мовою і країні	`Locale(String language, String country)`
Мовою, країні і варіанті мови	`Locale(String language, String country, String variant)`

Назва мов і країн повинні бути по стандартах:

- ISO-639 - мову,
- ISO-3166 - країна.

У той же час допустимо використовувати будь-які назви. Будь-які винятки такий підхід не викличе (але так робити не слід). Назва локалі визначається значеннями, які були передані в конструктор при її створенні:

```
Locale locale =
     new Locale("мова", "КРАЇНА", "варіант");
System.out.println (locale); // язик_СТРАНА_варіант
```

12.5 Код мови

Стандарт ISO 639 визначає коди більшості мов:

Найменування	опис
ISO 639-1 (1998)	2-буквені скорочення
ISO 639-2 (2002)	3-літерні скорочення

Приклади:

Мова	639-1	639-2
англійська	en	eng
Русский	ru	rus
Український	uk	ukr
французький	fr	fra

12.6 Код країни

ISO 3166 - міжнародний стандарт ISO, який визначає позначення держав і залежних територій, а також основних адміністративних утворень всередині держав.

приклади:

TR	Туреччина
UG	Уганда
UZ	Узбекистан
UA	Україна
UY	Уругвай

Стандарт складається з наступних частин:

Найменування	опис
ISO 3166-1	Коди держав і залежних територій
ISO 3166-1 alpha-2	двобуквених
ISO 3166-1 alpha-3	трьохбуквені
ISO 3166-1 numeric	цифрові
ISO 3166-2	Коди адміністративних утворень всередині держав (області, штати, провінції і т. П.)
ISO 3166-3	Коди вже неіснуючих держав (об'єднання, поділ, зміна назви і т. П.)

12.7 Локаль за замовчуванням

Поточний варіант регіональних налаштувань можна отримати за допомогою методу getDefault. Ці настройки визначає ОС в якій запущена JVM.

```
Locale locale = Locale.getDefault();
```

12.8 Пакети ресурсів

Для локалізації додатків створюють так звані пакети ресурсів (resource bundles). Кожен пакет являє собою файл властивостей або клас, який описує елементи, специфічні для конкретного регіонального стандарту (наприклад, повідомлення, написи і т.д.). Файл властивостей має розширення .properties

Імена пакетів ресурсів

призначення	шаблон
Для мови, країни і варіанти	ІМ'Я-ПАКЕТУ-РЕСУРСОВ_ЯЗИК_СТРАНА_ВАРІАНТ
Для мови і країни	ІМ'Я-ПАКЕТУ-РЕСУРСОВ_ЯЗИК_СТРАНА

для мови	ІМ'Я-ПАКЕТУ-РЕСУРСОВ_ЯЗИК
Без вказівки мови, країни і варіанти (в ньому - ресурси, що застосовуються за замовчуванням)	ІМ'Я-ПАКЕТУ-РЕСУРСІВ

Для завантаження пакета ресурсів використовують метод getBundle:

```
ResourceBundle bundle =
     ResourceBundle.getBundle("MyResources", locale);
```

Для отримання значення по ключу використовують метод getString ::

```
String value1 = bundle.getString("key1");
String value2 = bundle.getString("key2");
...
```

Якщо не вказати locale, то буде використана локаль за замовчуванням (її визначає JVM з налагодження операційної системи).

```
ResourceBundle bundle =
     ResourceBundle.getBundle("MyResources");
```

12.9 Алгоритм завантаження пакета ресурсу

Якщо при отриманні пакету ресурсів вказана локаль:

```
ResourceBundle bundle =
     ResourceBundle.getBundle("bundleName", locale);
```

яка була визначена так:

```
Locale locale = new Locale("L", "C", "V");
```

де

- L - код мови,
- C - код країни,
- V - варіант мови,

і, при цьому, локаль за замовчуванням (цю локаль створює сама JVM) визначена з використанням:

- коду мови DL;
- коду країни DC;
- варіанти мови DV.

Те при виклику методу getBundle пакет ресурсів буде завантажений з першого існуючого файлу:

1) bundleName_L_C_V.properties
2) bundleName_L_C.properties
3) bundleName_L.properties
4) bundleName_DL_DC_DV.properties
5) bundleName_DL_DC.properties
6) bundleName_DL.properties
7) bundleName.properties

Зауваження. Для завантаження пакетів ресурсів з класів алгоритм завантаження аналогічний щойно розглянутому, при цьому завантаження класу має пріоритет над завантаженням файлу властивостей.

12.10 Алгоритм пошуку ресурсу по ключу

Для завантаженого пакету ресурсів формується ланцюжок його предків (parents) шляхом послідовного відкидання елементів суфікса в імені пакету ресурсів. Далі відбувається пошук ресурсу по ключу починаючи з завантаженого пакету ресурсів і послідовно у всіх його предків, поки не буде знайдений заданий ключ.

У разі, якщо у всьому ланцюжку пакетів ресурсів ключ відсутній, буде викинуто виключення: MissingResourceException.

Приклад. Якщо завантажений пакет ресурсів

```
bundleName_L_C_V.properties
```

де L - код мови, C - код країни, V - варіант мови.

Те ланцюжок має наступний вигляд:

1) bundleName_L_C_V.properties
2) bundleName_L_C.properties
3) bundleName_L.properties
4) bundleName.properties

12.11 Файли властивостей

Для локалізації інтерфейсу додатку, зазвичай все рядки, які потрібно локалізувати, поміщають у відповідні файли властивостей.

Файл властивостей - текстовий файл, кожен рядок якого містить ключ і значення:

```
key1 = value1
key2 = value2
...
```

Кодування файлів властивостей: ISO-8859-1 (Latin-1). Це однобайтном кодування, яка є підмножиною Unicode кодування (перші 256 символів). Метод getString класу ResourseBundle читає файли властивостей використовуючи кодування ISO-8859-1.

Для запису символів, які не представлені цієї кодуванням (національні алфавіти, що відрізняються від латинського), слід використовувати java Unicode escape послідовності.

У стандартний набір поставки JDK (від Oracle) входить утиліта native2ascii призначена для спрощення створення файлів властивостей на національних мовах.

Приклади використання:

```
native2ascii -encoding Cp866 input output
```

в файл output будуть записані рядки файлу input у вигляді екранованих послідовностей; при цьому символи вихідного файлу будуть прочитані за допомогою кодування Cp866.

```
native2ascii input output
```

то ж що і в попередньому прикладі, але замість Cp866 буде використана кодування за замовчуванням.

12.12 Класи, що реалізують пакети ресурсів

Для підтримки ресурсів, які не є рядками, необхідно визначити класи, які є підкласами класу ResourceBundle. Вибір імен таких класів здійснюється відповідно до угод про іменування. Для завантаження класу використовується той же метод getBundle(), що і для завантаження властивостей.

Якщо два пакети ресурсів, один з яких реалізований у вигляді класу, а інший як файл властивостей мають однакові імена, то при завантаженні перевагу буде віддано класу.

12.13 Приклад 1

Локаль за замовчуванням: en_GB
Пакети ресурсів:

```
MyResources.class                   MyResources.properties
MyResources_fr.properties           MyResources_fr_CH.class
MyResources_fr_CH.properties
MyResources_en.properties           MyResources_es_ES.class
```

виклик

```
getBundle("MyResources", new Locale("fr", "CH"))
```

завантажить пакет

```
MyResources_fr_CH.class
```

при цьому, пошук значення по ключу буде вестися послідовно в:

```
1) MyResources_fr_CH.class
2) MyResources_fr.properties
3) MyResources.class
```

12.14 Приклад 2

Локаль за замовчуванням: en_GB
Пакети ресурсів:

```
MyResources.class                   MyResources.properties
MyResources_fr.properties           MyResources_fr_CH.class
MyResources_fr_CH.properties
MyResources_en.properties           MyResources_es_ES.class
```

виклик

```
getBundle("MyResources", new Locale("fr", "FR"))
```

завантажить пакет

```
MyResources_fr.class
```

при цьому, пошук значення по ключу буде вестися послідовно в:

```
1) MyResources_fr.properties
2) MyResources.class
```

12.15 Приклад 3

Локаль за замовчуванням: en_GB
Пакети ресурсів:

```
MyResources.class                   MyResources.properties
MyResources_fr.properties           MyResources_fr_CH.class
MyResources_fr_CH.properties
MyResources_en.properties           MyResources_es_ES.class
```

виклик

```
getBundle("MyResources", new Locale("ru", "RU"))
```

завантажить пакет

```
MyResources_en.class
```

при цьому, пошук значення по ключу буде вестися послідовно в:

```
1) MyResources_en.properties
2) MyResources.class
```

13 ПОТОКИ ВВЕДЕННЯ / ВИВЕДЕННЯ

13.1 Види потоків введення / виведення

Існує 2 види потоків введення / виведення:

- байтові;
- символьні.

Байтові потоки - сукупність електронних даних (byte). Символьні - послідовність символів BMP Unicode (char). Більшість потоків ядра Java (стандартного API) є нащадками 4-х суперкласів, які абстрактні і безпосередньо успадковані від класу Object.

суперклас ієрархії	*потоки*
`java.io.InputStream`	Вхідні байтові потоки
`java.io.OutputStream`	Вихідні байтові потоки
`java.io.Reader`	Вхідні символьні потоки
`java.io.Writer`	Вхідні символьні потоки

Зауваження. До складу API входить клас java.io.RandomAccessFile, який не належить до зазначених вище ієрархій, успадковується безпосередньо від класу Object і призначений для роботи з файлами, підтримуючи довільний доступ до їх вмісту.

Зауваження. Більшість потоків введення / виведення міститися в пакеті java.io, потоки для роботи з архівами міститися в пакеті java.util.

13.2 Парні потоки

Призначення кожного класу-потоку полягає в тому, щоб передати або прийняти послідовність символів або байт.

API Java містить більше 60 потоків, кожен з яких містить свій власний набір методів для управлінням процесом прийому / передачі інформації.

Для деяких потоків існують парні їм в тому сенсі, що парний потік містить дзеркальне відображення функціональності вихідного потоку щодо направлення передачі інформації.

Парні класи в ієрархіях байтових потоків

Input`Stream`	`OutputStream`
`ByteArrayInputStream`	`ByteArrayOutputStream`
`FileInputStream`	`FileOutputStream`
`StringBufferInputStream`	-
`ObjectInputStream`	`ObjectOutputStream`
`FilterInputStream`	`FilterOutputStream`

BufferedInputStream	BufferedOutputStream
-	PrintStream
ZipInputStream	ZipOutputStream
PushbackInputStream	-
DataInputStream	DataOutputStream

Парні класи в ієрархіях символьних потоків

Reader	*Writer*
BufferedReader	BufferedWriter
-	PrintWriter
StringReader	StringWriter
FilterReader	FilterWriter
PushbackReader	-
InputStreamReader	OutputStreamWriter
FileReader	FileWriter

13.3 Поле out класу System

Статичне поле out класу System має тип java.io.PrintStream, який являє собою надбудову над байтовим вихідним потоком OutputStream і за замовчуванням пов'язаний з консольним висновком (дисплеєм). Це, так званий, потік стандартного виведення. Програмно він може бути надбудований для того, щоб здійснювати перекодування символів даних, що виводяться.

13.4 Класи надбудови

Основне призначення надбудов - наділення існуючого потоку новими властивостями. API Java містить набір неабстрактне класів-надбудов, які є нащадками базових надбудов.

базовий клас	клас надбудова
InputStream	FilterInputStream
OutputStream	FilterOutputStream
Reader	FilterReader
Writer	FilterWriter

Комбінуючи вихідний потік і класи надбудови, можна створити новий потік із заданим набором властивостей.

Якщо потрібно наділити існуючий потік деяким властивістю, досить надбудувати його відповідним класом надбудовою і працювати з об'єктом останнього.

13.5 Клас DataInputStream

Клас DataInputStream успадковує клас надбудову FilterInputStream і дозволяє читати дані з вхідного байтового потоку в форматі примітивних типів даних: double, boolean і т.д.

Парний клас DataOutputStream успадковує клас FilterOutputStream і дозволяє записувати значення примітивних типів у вихідний байтовий потік, який потім можна буде прочитати використовуючи клас DataInputStream.

Зауваження. Примірники класів DataInputStream і DataOutputStream надбудовують, відповідно, вхідний і вихідний потоки, які передаються їм як параметри конструкторів при їх створенні.

13.6 Клас BufferedOutputStream

Клас BufferedOutputStream успадковує клас надбудову FilterOutputStream. Об'єкт цього класу надбудовує вихідний байтовий потік і підтримує буфер певного розміру.

Вихідний потік і розмір буфера передаються об'єкту BufferedOutputStream при його створенні за допомогою конструктора в якості параметрів (розмір буфера за замовчуванням як правило достатній для вирішення більшості виникаючих завдань).

Парний клас BufferedInputStream успадковує надбудову FilterInputStream і надбудовує вхідний потік, додаючи можливість використовувати буфер.

13.7 Клас ByteArrayInputStream

Клас ByteArrayInputStream успадковується безпосередньо від класу InputStream, при цьому байти зчитуються в масив байт, який передається конструктору об'єкта класу ByteArrayInputStream при його створенні.

Парний клас ByteArrayOutputStream успадковується безпосередньо від класу OutputStream, при цьому байти записуються в масив байт, який передається конструктору об'єкта класу ByteArrayOutputStream при його створенні.

13.8 Клас FileOutputStream

Клас FileOutputStream успадковується безпосередньо від класу OutputStream і призначений для запису байт в файл, ім'я файлу передається конструктору при створенні об'єкта.

Парний клас FileInputStream успадковується безпосередньо від класу InputStream і призначений для читання байт з файлу, ім'я файлу передається конструктору при створенні об'єкта.

13.9 Клас PushbackInputStream

Клас PushbackInputStream надбудовує вхідний байтовий потік і дозволяє крім читання здійснювати запис прочитаних байт назад у вхідний потік.

Зауваження. Клас PushbackInputStream не має парний клас. Існує аналогічний клас для вхідних символьних потоків.

13.10 Клас RandomAccessFile

Для створення об'єктів класу RandomAccessFile потрібно передати конструктору класу ім'я файлу, з яким передбачається працювати, а також обов'язково режим доступу до файлу:

- 'R' (тільки читання);
- 'Rw' (читання і запис).

Замість імені файлу можна передати відповідний об'єкт класу File.

Зауваження. Відсутня режим доступу "тільки запис".

13.11 Клас OutputStreamWriter

Клас OutputStreamWriter успадковується від класу Writer, і перетворює вихідний символьний потік у вихідний байтовий потік. Клас має декілька конструкторів, кожен з яких приймає в якості одного зі своїх параметрів вихідний символьний потік.

- OutputStreamWriter (OutputStream out)
- OutputStreamWriter (OutputStream out, String charsetName)

Другий параметр вказує на кодування, при цьому кожному символу ставиться у відповідність сукупність байт, яка є числовим кодом символу в цьому кодуванні.

Зауваження. Якщо при створенні об'єкта класу OutputStreamWriter використовується конструктор без вказівки кодування, то конвертація здійснюється з використанням кодування за замовчуванням.

13.12 Кодування за замовчуванням

При запуску програми кодування за замовчуванням встановлює JVM в залежності від операційної системи в якій виконується програма і її налаштувань. ОС Windows використовує в якості кодування за замовчуванням Windows-1251 (Cp1251), для виведення в консоль використовується DOS-кодування Cp866 (Win OS російської локалізації).

Зауваження. Якщо при створенні об'єкта класу OutputStreamWriter був використаний конструктор без вказівки кодування, то конвертація буде здійснене з використанням кодування за замовчуванням.

13.13 Вказівка кодування при компіляції

Для правильного відображення строкових літералів, записаних в програмі, слід забезпечити правильне конвертування цих символів в Unicode при компіляції за допомогою javac, вказавши це за допомогою ключа -encoding.

Наприклад, якщо код програми записаний в DOS кодуванні Cp866, то компілювати необхідно так:

```
javac -encoding Cp866 NameOfJavaFile
```

13.14 Перекодування виведення

Всі рядкові літерали в байт коді класів представлені в кодуванні Unicode в форматі UTF-16BE.

При виведенні таких рядків на екран, у файл і т.д. здійснюється їх перекодування з використанням кодування за замовчуванням.

Наприклад, в ОС Windows кодуванням за замовчуванням є Cp1251, тому станеться конвертування

```
Unicode⇒ Cp1251
```

Якщо висновок здійснюється в консольне вікно (за допомогою методу System.out.println), то такі рядки в загальному випадку будуть відображені неправильно, тому що Windows для відображення символів у командному вікні використовує кодування Cp866.

Щоб уникнути цього, необхідно явно вказати в якому кодуванні повинні виводиться символи. Досягається це за допомогою надбудови стандартного потоку виводу:

```
PrintWriter out = new PrintWriter (
     new OutputStreamWriter (System.out, "Cp866"), true);
out.println (s); // вивід на екран рядка s в кодуванні Cp866
```

Зауваження. Другий параметр конструктора PrintWriter вказує на те, що кожен виклик методу println буде примусово скидати буфер, тобто, після кожного виклику println відбуватиметься висновок на екран строкового значення параметра цього методу. В іншому випадку висновок на екран відбудеться тільки тоді, коли буфер примусово буде скинутий за допомогою виклику методу flush.

Зауваження. Аналогічно можна надбудувати по суті будь-який потік, таким чином досягається можливість здійснювати перекодування символів між будь-якими двома допустимими кодуваннями.

13.15 Кодування за замовчуванням

Рядок з ім'ям кодування за замовчуванням міститься в системній властивості file.encoding. Властивість можна програмно змінити, проте не для всіх JDK це призведе до дійсної зміни кодування за замовчуванням.

```
String encoding, s = "абвгд";
encoding = System.getProperty("file.encoding");

// encoding = Windows-1251
System.setProperty("file.encoding", "Cp866");

encoding = System.getProperty("file.encoding");
// encoding = Cp866

// вивід в кодуванні Windows-1251 (jdk1.4 / 5.0):
```

```
System.out.println (s);
```

13.16 Клас InputStreamReader

Клас InputStreamReader успадковується від класу Reader, і перетворює вхідний байтовий потік в символьний використовуючи задану кодування.

- InputStreamReader (InputStream in)
- InputStreamReader (InputStream in, String charsetName)

13.17 Буферизація

Для прискорення файлових операцій читання / запису слід використовувати Буферізірованний класи: BufferedInputStream і BufferedReader.

```
BufferedReader in1 = new BufferedReader (
     new InputStreamReader (new FileInputStream("file.txt")));

BufferedReader in2 = new BufferedReader (
     new FileReader("file.txt"));

BufferedInputStream in3 = new BufferedInputStream (
     new FileInputStream("file.txt"));
```

13.18 Поле in класу System

Статичне поле in класу System має тип InputStream і пов'язано за замовчуванням з консольним введенням (клавіатурою). Як правило, доводиться надбудовувати цей вхідний потік.

```
BufferedReader in = new BufferedReader (
     new InputStreamReader (System.in));

String s = null;
while (! (s = in.readLine()). equals("")) {
     System.out.println (s);
}
```

13.19 Момент створення файлу

Фізичне створення файлу за допомогою класу FileOutputStream відбувається до того, як закінчиться виконуватися конструктор, за допомогою якого створювався відповідний об'єкт.

```
File f = new File("file");

// буде створений файл "file"
FileOutputStream out = new FileOutputStream (f);
```

14 КЛАС FILE

Клас File призначений для роботи з елементами файлової системи (ЕФС). Подання шляху до ЕФС залежить від операційної системи. Об'єкт класу File є

абстрактне уявлення шляху до ЕФС (файлу або каталогу), при цьому сам ЕФС може бути відсутнім.

При створенні об'єкта класу File завжди задають абстрактний шлях до ЕФС.

```
public File (String pathname) {...}
```

Шлях до ЕФС може бути заданий шляхом складання двох шляхів:

```
public File (String parent, String child)
public File (File parent, String child)
```

14.1 Абстрактний шлях

Абстрактний шлях складається з необов'язкового системно-залежного префікса і послідовності імен. В ОС Windows префіксом є ім'я пристрою (hdd, CD-drive і т.п.), на якому розташована файлова система, \і \\. Послідовність імен складається із сукупності імен каталогів, причому останнім ім'ям послідовності може бути ім'я файлу.

Імена поділяються роздільником, прийнятим в ОС в якій виконується JVM. Сам роздільник зберігається в системній властивості file.separator, а також в статичному полі separator класу File.

```
File f = new File("folder" + File.separator + "file");
```

Зауваження. В ОС Windows роздільником є символ \ (зворотний слеш, backslash).

При написанні строкового літерала, що містить шлях до ЕФС, слід цей символ подвоювати:

```
File f = new File("folder\\file");
```

Також в ОС Windows (останні версії) допустимо роздільник / (slash).

```
File f = new File("folder / file");
```

При створенні об'єкта File допускається вказувати порожній абстрактний шлях.

```
File f = new File("");
```

14.2 Метод getPath

Метод getPath класу File повертає строкове представлення абстрактного шляху до ЕФС, який був заданий при створенні об'єкта File, при цьому використовується роздільник файлової системи ОС, в якій виконується JVM.

Метод toString класу File повертає рядок, яку генерує метод getPath.

```
// якщо виконується в Windows
File f2 = new File("folder/file");

// то буде виведено folder\file
System.out.println (f2);
```

14.3 Перетворення абстрактного шляху

Абстрактний шлях, який задається об'єкту File при його створенні, перетворюється в рядок за такими правилами:

1) роздільники за останнім ім'ям відкидаються;

2) кілька поспіль роздільників всередині шляху будуть замінені на один;

3) кілька поспіль роздільників на початку шляху інтерпретуються в залежності від ОС, в Windows вони будуть замінені на два роздільник.

```
File f = new File("\\/file//");
System.out.print(f); // буде виведено \\ file
```

14.4 Метод getAbsolutePath

Метод getAbsolutePath класу File повертає т.зв. абсолютний шлях до ЕФС вид якого залежить від ОС. У Windows можливо два випадки:

1) абстрактний шлях не містить префікса: в цьому випадку метод поверне рядок, що складається з шляху до поточного призначеному для користувача каталогу, роздільник і абстрактного шляху;

2) абстрактний шлях містить префікс, в цьому випадку метод поверне абстрактний шлях.

Зауваження. Поточний призначений для користувача каталог призначається кожній програмі при її виконанні і призначений для вирішення відносних шляхів, його ім'я міститься в системній властивості user.dir

```
String s;
File f = new File("path \\ file");

s = f.getAbsolutePath());
// s ⇒ System.getProperty("user.dir") + "\\ path \\ file"

f = new File("Z: \\ path / file");
s = f.getAbsolutePath());
// s ⇒"Z: \\ path \\ file"
```

Зауваження. Якщо абстрактний шлях порожньої, то буде виведено значення властивості user.dir.

Зауваження. Рядок, яку повертає метод getAbsolutePath об'єкта File, не залежить від того, існує чи ні ЕФС на який посилається цей об'єкт.

14.5 Метод listFiles

Метод listFiles класу File повертає масив об'єктів File, які поставлені в відповідності ЕФС, що містяться в каталозі, на який вказує об'єкт this.

```
public File [] listFiles()
```

Можливі три випадки:

1) this посилається на непорожній каталог: повертається масив буде включати об'єкти File, відповідні елементам файлової системи, які містяться в цьому каталозі;

2) this посилається на порожній каталог: повертається масив буде мати нульову довжину;

3) this посилається на файл (НЕ каталог), в цьому випадку метод listFiles поверне значення null.

Зауваження. Виклик методу listFiles бажано випереджати викликом методу isDirectory, який повертає true в тому і тільки в тому випадку, коли об'єкт File, на якому він викликається, посилається на існуючий каталог.

```
File [] list;
if (file.isDirectory()) list = file.listFiles();
```

14.6 Інтерфейс FileFilter

Метод listFiles може приймати в якості параметра об'єкт класу, який реалізує інтерфейс FileFilter. Цей інтерфейс містить один метод accept.

```
boolean accept (File pathname)
```

У результуючий список, який повертає listFiles будуть включені ті і тільки ті об'єкти File, для яких метод accept повертає true.

```
import java.io. *;

public class Test implements FileFilter {
     public boolean accept (File f) {
          return f.getName(). length() <8;
     }
     public static void main (String [] argv) {
          File path = new File("");
          File [] list = path.listFiles (this);
     }
}
// list містить тільки ті ЕФС з каталогу user.dir
// імена яких мають довжину чиєму імені менше 8 символів
```

14.7 Метод getParent

Метод getParent класу File повертає частину абстрактного шляху ЕФС.

Структура абстрактного шляху	Значення, що повертається getParent
тільки одне ім'я	null
тільки одне ім'я, попереджання роздільником	залежить від OS, для Windows - роздільник
тільки одне ім'я, попереджання двома роздільниками	залежить від OS, для Windows - два роздільник

кілька імен	абстрактний шлях без останнього імені і роздільник, який його випереджає

14.8 Метод getCanonicalPath

Метод getCanonicalPath класу File повертає т.зв. канонічний шлях до ЕФС. Даний метод намагається визначити реальний шлях до ЕФС в файлової системі ОС, на якій виконується JVM. Процес визначення залежить від ОС. Для Windows він полягає в наступному:

1) за допомогою методу getAbsolutePath визначається абсолютний шлях;

2) відповідним чином обробляються імена

- . - поточний каталог;
- .. - (батьківський каталог);

3) контакт передачі (якщо воно є в дорозі) наводиться до верхнього регістру.

```
File f = new File("c: / java / test");
File f2 = new File (f, "..");
String s = f2.getCanonicalPath());

// s ⇒ c: \ java
```

15 ВИНЯТКИ

Виняток - це ситуація, що перешкоджає подальшому нормальному виконанню програми.

Виняток в Java - це об'єкт, який описує аварійну ситуацію, що сталася в деякій частині коду програми.

Обробка винятків здійснюється за допомогою спеціальної конструкції try / catch / finally.

15.1 Ієрархія винятків

Класи винятків складають ієрархію, на верхньому рівні якої знаходиться суперклас всіх винятків Throwable.

```
public class Throwable extends Object implements Serializable
```

ієрархія:

- Throwable - суперклас всіх винятків
  - Error - предок винятків JVM, зовнішніх до додатка
  - Exception - предок всіх програмних винятків
    - RuntimeException - наслідок багів або порушення контрактів.

15.2 Перевіряються і непроверяемие виключення

Всі винятки можуть бути розділені на дві категорії:

- перевіряються винятку (checked exceptions);
- непроверяемие виключення (unchecked exceptions).

Перевіряються виключення, це виключення, які в обов'язковому порядку вимагають або обробки або вказівки в секції throws методу.

Непроверяемие виключення можна не обробляти і можна не вказувати в секції throws.

виняток	Тип
Throwable	проверяемое
Error	непроверяемую
Exception	проверяемое
RuntimeException	непроверяемую

Зауваження. Властивість виключення бути перевіряється або непроверяемим передається при спадкуванні класів, за винятком бібліотечних класів Error і RuntimeException - обидва цих класу є непроверяемимі винятками, успадкованими від перевірених.

15.3 Деякі класи з ієрархії виключень

15.3.1. Клас Throwable

Є суперкласом всіх виняткових ситуацій, містить набір публічних методів, які успадкованого усіма винятками. Є підприємством, що перевіряється винятком. Якщо його вказати в блоці catch, то цей блок відловив будь виняток, яке відбулося.

15.3.2. Клас Exception

Предок всіх програмних виняткових ситуацій. Проверяемое виняток. Як правило, для створення своїх власних виключень в якості базового класу вказують саме цей клас. При виникненні винятків даного типу, як правило, після їх обробки, додаток здатний продовжити свою роботу. Є підприємством, що перевіряється винятком.

15.3.3. Клас Error

Виняткові ситуації віртуальної машини. Це зовнішні по відношенню до додатка виключення. Відловлювати їх як правило не слід внаслідок того, що часто продовжувати виконання програми після їх виникнення недоцільно (закінчилася пам'ять, був завантажений не той клас, який потрібен, переповнення стека і т.п.). Є непроверяемим винятком.

15.3.4. Клас RuntimeException

Є нащадком класу Exception, причому це непроверяемую виняток. Після виникнення такого роду винятків додаток, якщо вони будуть виловлені і оброблені, як правило, додаток здатний продовжити свою роботу.

15.3.5. Клас NullPointerException

Виникає, якщо на об'єктній змінній викликати метод або звернутися до поля об'єкта, в разі, коду змінна посилається на null. Є спадкоємцем RuntimeException.

15.4 Ієрархія винятків

Всі винятки можуть бути спіймані за допомогою спеціальної секції catch (т.зв. блок обробки винятків). При цьому слід враховувати наступне:

- не слід обробляти виключення які є нащадками Error (хоча це можна зробити);
- можна не обробляти виключення, які є нащадками RuntimeException.

15.5 Дії JVM при виникненні виняткової ситуації

Ланцюжок викликів, починаючи від методу main до методу, де відбулося ісключніе називають стеком викликів (call stack).

Коли відбувається виняткова ситуація, JVM:

1) створює об'єкт виключення;

2) шукає обробник виключення, який може обробити виняток.

Якщо виключення відбулося усередині блоку try і воно може бути приведене до типу селектора однієї з пов'язаних секцій catch, то JVM передасть управління відповідної секції.

В іншому випадку, весь код методу, який слід за місцем, в якому відбулося виключення, буде знятий з виконання і JVM буде шукати секцію catch з відповідним селектором вгору по стеку викликів.

Якщо JVM не знайде обробник для виключення, то вона обробить виняток сама. За замовчуванням, друкує стек викликів, які призвели до виключення, в стандартний потік помилок.

Зауваження. Існує можливість перекрити стандартний обробник необроблених винятків.

15.6 Оператор throw

Викид виключення може бути здійснений примусово за допомогою оператора throw.

```
try {
     throw new NoSuchMethodException();
} Catch (Throwable t) {
     /*...*/
}
```

Відразу за ключовим словом throw повинен стояти об'єкт-виняток, який можна створити за допомогою відповідного конструктора або ж взяти готовий, наприклад, селектор блоку catch.

```
// метод може викинути виняток типу Exception
public void method throws Exception {
     try {
          // всередині нього викидаючием
          // виняток Exception
          throw new Exception();
     } Catch (Exception e) {// відловлюємо
          throw e; // викидаємо заново
     }
}
```

15.7 Блок try / catch / finally

За охоронюваним ділянкою коду try завжди повинен знаходиться один або кілька блоків обробки винятків catch або блок finally, який виконується завжди, в незалежності від того було або не було викинуто виключення в блоці try.

секція	опис
try	захищений ділянку
catch	блок обробки виключень
finally	блок звільнення ресурсів

Існує три можливих поєднання блоків try, catch і finally:

- try / catch
- try / catch / finally
- try / finally

Секцій catch може бути кілька.

Зауваження. Починаючи з Java 7, в мові існує конструкція try-with-resources, її можна використовувати без наступних за нею секцій catch і finally.

15.8 Секція catch

Є блоком обробки винятків. Приклад використання:

```
try {
... // викид виключення ex
} Catch (ExceptionType ex) {
...
}

// Попадаємо в блок catch якщо ex приводимо до ExceptionType
// тобто допустимо присвоювання:
// ExceptionType e = ex
```

15.9 Виконання коду, наступного за блоком обробки винятків

Як тільки в блоці try відбувається викид виключення, решта коду блоку try буде знята з виконання і управління буде передано відповідним блоку catch.

Якщо в блоці catch не відбувається викид виключення і не відбувається вихід з методу за допомогою оператора return, то наступного виконуваної інструкцією програми буде перший рядок за блоком catch.

```
boolean f = false;
String s = null;
try {
      // викид виключення
     // яке має тип NullPointerException:
     int y = s.length();

     f = true; // не буде виконано

// обробка виключення:
} Catch (NullPointerException e) {
     f = true; // буде виконано
}
f = true; // буде виконано
```

15.10 Блок finally

Якщо за блоком try / catch слід блок finally, то його вміст виконається завжди в незалежності від того, відбудуться чи ні викиди винятків в блоках try і catch.

```
try {
     throw new Exception();
} Catch (Exception e) {
      // вихід з блоку catch по виключенню:
     throw new Exception();
} Finally {
      // перед виходом виконається тіло блоку finally
     System.out.println("finally block is always executed");
```

```
}
```

Зауваження, якщо в секції catch прописати виклик завершення віртуальної машини, то секція finally виконана не буде, як втім і весь код програми (воно просто цілком буде знято з виконання).

```
System.exit (0);
```

15.11 Два способи реакції на виключення

Можливі два випадки:

1) метод може обробити виняток сам;

2) метод може покласти обробку виключення на зовнішній по відношенню до нього код.

Розглянемо їх детальніше.

1) Метод може обробити виняток сам за допомогою конструкції try / catch, при цьому джерело виключення повинен знаходиться всередині захищеного ділянки try, а блок catch повинен мати в якості селектора відповідний тип виключення

```
public void method() {
     try {
          // викид виключення:
          throw new NoSuchMethodException();
     } Catch (Exception e) {
          // виняток буде спіймано, тому що
          // NoSuchMethodException розширює Ecxeption
     }
}
```

2) Метод може покласти обробку виключення на зовнішній по відношенню до нього код, випустивши виняток назовні; при цьому метод зобов'язаний задекларувати це за допомогою ключового слова throws і імені відповідного типу виключення.

Якщо метод задекларував за допомогою throws секції виключення деякого типу, то це означає, що він потенційно може викинути виключення такого типу або будь-якого його потоку.

```
try {
     method();
} Catch (Exception e) {
     /*...*/
}
...
// метод method може викинути виняток Exception
// або будь-якого його нащадка:
public void method() throws Exception {
     // викид виключення NoSuchMethodException,
```

```
     // яке успадковує Exception:
     throw new NoSuchMethodException();
}
```

15.12 Ключове слово throws

Якщо при оголошенні методу в секції throws вказується ім'я класу винятку, то це означає, що метод є потенційним джерелом винятків цього типу (або будь-яких його нащадків), але не означає, що метод обов'язково викине виняток.

Виклик такого методу повинен бути укладений в блок try / catch або ж метод, який містить даний виклик повинен задекларувати потенційний викид відповідного виключення в секції throws.

Зауваження. Головний метод програми main може мати секцію throws, при цьому винятки, які він може викинути буде обробляти JVM.

16 ДЕЯКІ ВЛАСТИВОСТІ ВИНЯТКІВ

16.1 Множинний селектор секції catch

Починаючи з Java 7 через знак | (Вертикальна риса, pipe) можна вказувати відразу кілька селекторів у одній catch секції (т.зв. multicatch).

```
try {
...
} Catch (Type1 | Type2 | Type3 ex) {
...
}
```

Зауваження. Змінна, яка стоїть праворуч від селекторів неявно фіналізована (тобто її значення не можна змінити всередині catch секції).

```
try {
...
} Catch (Type ex) {
     // no error
     ex = null;
} Catch (Type | Type2 | Type3 ex) {
     // error!
     ex = null;
}
```

16.2 Викид виключення в блоці catch

Якщо в блоці catch відбувається викид виключення, то його обробкою не займатимуться наступні за даними блоки catch.

```
try {
     throw new NullPointerException();
} Catch (NullPointerException e) {
     // вихід з блоку try / catch:
     throw new Exception();
```

```
} Catch (Exception e) {
     // даний блок catch не буде виконано
}
```

16.3 Оператор return в блоці catch

Якщо в catch блоці відбувається передача керування за допомогою оператора return, то значення його операнда запам'ятовується, виконується вміст блоку finally (якщо він є), після чого відбувається виконання оператора return, якому в якості операнда буде передано збережене значення.

Таким чином, виконання вмісту блоку finally не впливає на повертається оператором return значення.

```
public int getX() {
     int x = 1;
     try {
          throw new Exception();
     } Catch (Exception ex) {
          return x = 2;
          // значення 2 запам'ятовується
     } Finally {
          // буде виконано перед виходом з методу
          x = 3;
          // x ⇒ 3
     }
}
...
int x = getX();
// метод поверне збережене значення 2
// x ⇒ 2
```

Зауваження. Якщо finally блок містить оператор return, то вихід з try / catch / finally блоку буде відбуватися по оператору return, який міститься в finally блоці.

```
public int getX() {
     int x = 1;
     try {throw new Exception();}
     catch (Exception ex) {
          return x = 2;
     } Finally {
          return x = 3;
     }
}
...
int x = getX();
// збережене значення 2 буде проігноровано:
// x ⇒ 3
```

16.4 Придушення виключення при виході з return в finally

Якщо в захищеному блоці сталося виняток, а в finally секції варто оператор управління return, то викинута виняток буде придушене.

```
static void m() {
     try {
          // викид виключення ArithmeticException
          int x = 3/0;
     } Finally {
          return;
     }
}

public static void main (String [] args) {
     m();
     System.out.println("ok");
     // ok
}
```

16.5 Виключення при перекритті методу

При перекритті методу не може бути розширено безліч тих винятків, які задекларовані в секції throws.

```
class A {
     void m() throws IOException {}
}

class B extends A {
     // помилка компіляції, розширено безліч
     // викидаються винятків:
     void m() throws Exception {}
}
```

16.6 Пропуск назовні виключення всередині іншого виключення

Якщо перекривається в класі нащадку метод потенційно може викинути виключення, які не задекларовано в секції throws методу класу предка, то слід:

1) перехопити такі винятки;

2) помістити їх усередину якогось винятку з ієрархії:

   a) RuntimeException,

   b) Error;

3) примусово його викинути;

4) виклик методу помістити в блок try / catch, в якому витягти вихідне виняток за допомогою методу getCause

```
public Throwable getCause()
```

Приклад:

```
import java.io.IOException;

class A {
     // викидає тільки Error або RuntimeException
     void m() {}
}

class B extends A {
     void m() {
          try {
               throw new IOException();
          } catch (IOException ex) {
               // викид нового виключення
               throw new RuntimeException (ex);
          }
     }
}
...
B b = new B();
try {
     // метод викине виняток RuntimeException:
     b.m();
} Catch (RuntimeException t) {
     // витяг вихідного виключення
     Throwable cause = t.getCause();

     // друкує стек вихідного виключення
     cause.printStrackTrace();
}
```

16.7 Конструкція try-with-resources

Починаючи з Java 7 в мові з'явилася конструкція try-with-resources, за допомогою якої можна домогтися автоматичного закриття відкритих ресурсів. Під ресурсами розуміють об'єкти класів, які реалізують інтерфейс

```
package java.lang;
public interface AutoClosable {
     void close() throws Exception;
}
```

При виході із секції try JVM здійснить спробу закриття ресурсів, які були оголошені в заголовку try:

```
try (Scanner s = new Scanner (fileName)) {
     // ...
}
// при виході з секції try буде викликаний метод s.close()
```

Зауваження. Починаючи з Java 7 інтефейс

```
package java.io;
public interface Closable extends AutoCloseable {
     void close() throws IOException;
```

```
}
```

успадковує інтерфейс AutoCloseable.

Згідно зі специфікацією мови Java, конструкція try-with-resources є скороченим записом вкладених конструкцій звичайних try / catch / finally.

16.8 Секція catch конструкції try-with-resources

Секція catch здатна перехопити не тільки виключення, які відбуваються в секції try, а й ті, які може викинути метод close оголошених в заголовку try ресурсів (зрозуміло, коли ці вилучення узгоджені за типом з селектором catch). Іншими словами, секція catch діє як обробник виключень не тільки для секції try, але і для методів close всіх ресурсів, які задекларовані.

У разі якщо виключення відбувається в try секції і в методі close хоча б одного з ресурсів, то захід у секцію catch іде за рахунок першому виключенню (то, яке відбулося в try), а всі інші придушуються і їх можна згодом отримати за допомогою виклику методу Throwable #getSuppressed:

```
Throwable [] getSuppressed()
```

16.9 Придушення викиду винятків в конструкції try-with-resources

Покладемо, що заголовок секції try містить оголошення одного AutoCloseable ресурсу. Якщо викид виключення відбувається в секції try, то перед тим, як управління буде передано секції catch з відповідним селектором (або виключення "вийде назовні") буде зроблений виклик методу close на ресурсі. Якщо close викидає виключення, то цей викид буде пригнічений, а саме виключення буде приєднано як "suppressed" виключення до виключення, з яким завершується секція try.

Приклад:

```
try (AutoCloseable r = ...) {
...
} Catch (ExceptionType ex) {
...
}
```

Варіанти, які можуть статися за умови, що виключення R і T приводили до селектору ExceptionType:

виняток в		
try	r.close	результат
-	-	Виконання try, потім r.close
-	R	catch обробляє R
T	-	catch обробляє T
T	R	catch обробляє T, T містить посилання на R як на пригнічений виняток (suppressed)

16.10 Власні класи виключень

Свої власні класи виключень можна створювати шляхом успадкування від будь-якого класу з ієрархії Throwable.

Зокрема, наприклад, в якості контейнера для винятків в попередньому пункті можна використовувати своє власне виключення:

```
class ExceptionHolder extends RuntimeException {...}
```

16.11 Методи toString і getMessage класу Throwable

Клас Throwable містить методи toString і getMessage.

```
public String toString()
public String getMessage()
```

які успадковуються всіма класами винятками. Метод getMessage повертає опис винятку. Метод toString повертає рядок, складений з імені класу винятку, двокрапки з пропуском і рядки, яку повертає метод getMessage.

При визначенні призначеного для користувача класу винятків можна перекрити обидва цих методу. Однак, як правило, перекривають тільки метод getMessage.

17 ПОТОКИ ВИКОНАННЯ

17.1 Виконання інструкцій потоками

Потік виконання - послідовність команд, які виконуються процесором. Інші назви: потік обчислення, нитка, (англ.) Thread.

Виконувана програма може мати декілька потоків. Будь-яку інструкцію (виклик методу, оператор, операція і т.п.) завжди виконує деякий потік.

17.2 Суперклас потоків виконання

Суперкласом всіх потоків виконання є клас

```
java.lang.Thread
```

17.3 Потік як об'єкт

Вираз "на потоці викликаний метод" слід розуміти як виклик методу на об'єкті відповідного класу потоку.

17.4 Потік як процес виконання команд

Вираз «потік виконує метод» слід розуміти як виконання інструкцій методу потоком.

17.5 Головний потік

Будь-яка програма має хоча б один потік обчислень - головний потік. Точкою входу в головний потік є метод main.

```
public class test {
     public static void main (String [] argv) {
          // інструкції методу main
          // виконує головний потік програми
     }
}
```

Головний потік запускає JVM. Всі інструкції методу main виконує головний потік.

17.6 4. Статичні методи класу Thread

У будь-якому місці програми можна викликати статичні методи класу Thread, які відносяться до поточного потоку, тобто до того потоку, який викликає ці методи.

Наприклад, посилання на об'єкт Thread поточного потоку можна отримати за допомогою статичного методу

```
Thread.currentThread()
```

17.7 Створення та запуск дочірнього потоку (extends Thread)

Щоб створити потік можна розширити клас Thread, перекривши його метод run

```
class B extends Thread {
     public void run() {
          // do something
     }
}
```

Після створення потік можна запустити на виконання за допомогою методу start класу Thread.

```
B b = new B();
b.start();
// або
new B().start();
```

зауваження:

- run - точка входу в потік
- main - точка входу в головний потік (головний потік запускає JVM).

17.8 Створення і запуск дочірнього потоку (implements Runnable)

Інший спосіб створення потоку полягає в реалізації інтерфейсу Runnable, який має єдиний метод run.

```
class B implements Runnable {
     public void run() {
          // do something
     }
}
```

Після цього створюють новий потік за допомогою конструктора Thread (Runnable target).

```
Thread t = new Thread (new B());
t.start();
// або
new Thread (new B()).start();
```

Зауваження. Клас Thread реалізує інтерфейс Runnable.

```
public class Thread implements Runnable
```

Зауваження. Як параметр конструктору Thread може бути переданий об'єкт класу, який успадковує Thread.

```
public class Test {
     public static void main (String [] argv) {
          Thread t = new Thread (
               new MyThread());
          t.start();
     }
}
class MyThread extends Thread {
```

```
     public void run() {}
}
```

17.9 8. Створення і запуск потоку в одному класі

Потік можна створити і запустити в одному класі.

```
public class MyThread extends Thread {
     public void run() {
          ...
     }

     public static void main (String [] argv) {
          // запуск потоку
          new MyThread().start();
     }
}
```

17.10 Запуск потоку в конструкторі класу-потоку

Потік можна запустити в конструкторі потоку.

```
class MyThread extends Thread {
     public MyThread() {
          this.start();
     }
     public void run() {
           // do something
     }
     public static void main (String [] argv) {
          new MyThread();
     }
}
```

17.11 Створення і запуск потоку за допомогою анонімного класу

Потік можна створити і запустити в методі за допомогою анонімного класу.

```
public class MyThread {
     public static void main (String [] argv) {
          new Thread() {
               public void run() {
                     // do something
               }
          } .start();
     }
}
```

17.12 Завершення виконання потоку

Потік починає своє виконання, коли на ньому викличуть метод start в батьківському потоці. Метод start в свою чергу викликає метод run.

Потік завершує своє виконання після виконання останньої інструкції методу run. Можливий вихід з потоків в зв'язку з викидом виключення або ж по оператору return. Аналогія для головного потоку - головний потік завершує своє виконання після виконання останньої інструкції методу main.

Зауваження. Існують т.зв. «Потоки-демони», які призначені для обслуговування інших потоків.

Якщо в програмі запущеними залишаються тільки потоки-демони, то JVM примусово припиняє їх роботу і завершує виконання програми.

Щоб зробити потік «демоном» необхідно перед його запуском викликати на ньому метод setDaemon, передавши значення true.

```
public class test extends Thread {
     public void run() { while (true); } //безкінечний цикл
     public static void main (String [] argv)
               throws InterruptedException {
          // головний потік створює потік test
          test t = new test();
          t.setDaemon (true); //робить його «демоном»
          t.start(); // запускає
          // щоб дочірній потік встиг запуститися:
          Thread.sleep (1000);
     } // в даному місці головний потік завершує своє
       // виконання і JVM завершує роботу
      // додатки перериваючи нескінченний цикл
}
```

17.13 Метод sleep класу Thread

Клас Thread містить статичний метод sleep, який робить паузу у виконанні поточного потоку на задане число мілісекунд.

```
public static void sleep (long millis)
          throws InterruptedException
```

Метод викине виняток InterruptedException, якщо на потоці для якого він робить паузу викликаний метод interrupt.

```
public static void main (String [] argv) {
     for (int j = 0; j <10; j ++) {
          System.out.println (j);
          // пауза головного потоку в 1 секунду
          try {
               Thread.sleep (1000);
          }
          catch (Exception e) {e.printStackTrace();}
     }
}

public class test extends Thread {
     public void run() {
          for (int j = 0; j <10; j ++) {
               System.out.println (j);
               try {Thread.sleep (200);}
```

```
                catch (Exception e) {e.printStackTrace();}
          }
     }
     public static void main (String [] args)
                    throws InterruptedException {
          test t = new test(); t.start();
          Thread.sleep (1000);
          t.interrupt();
          // через секунду буде викид
          // виключення методом sleep якщо він
     }    // виконується в даний момент
}
```

Зауваження. Метод sleep перевантажений. Точність часу паузи не гарантовано.

```
// пауза на ms мілісекунд
public static void sleep (long ms)

 // пауза на ms мілісекунд і ns наносекунд
public static void sleep (long ms, int ns)
```

Зауваження. InterruptedException є перевіряється винятком.

17.14 Метод alive класу Thread

Метод isAlive класу Thread повертає true, якщо потік, на якому він викликаний, запущений і ще не припинив своє виконання.

```
public class test extends Thread {
     public static void main (String [] argv) {
          // гл. потік створює і запускає новий потік
          test t = new test();
          t.start();
          // головний потік викликає метод
          // isAlive на занедбаному потоці:
          while (boolean isAlive = t.isAlive()) {
                // буде виводиться true поки
               // виконується запущений потік:
               System.out.println (isAlive);
          }    // число циклів заздалегідь невідомо
     }         // залежить від того, як довго буде
               // виконуватися метод run
     public void run() {}
}
```

17.15 Приклад запуску двох потоків, які виводять різні повідомлення з різною періодичністю

```
public class test extends Thread {
     private String mes;
     private int ms;
```

```
     // конструктор потоку
     public test (String mes, int ms) {
          this.mes = mes;
          this.ms = ms;
          // запуск потоку в конструкторі
          this.start();
     }

     public void run() {
          while (true) {// безкінечний цикл
               // висновок повідомлення
               System.out.println (this.mes);
               // пауза на ms мілісекунд
               try {sleep (this.ms);}
               catch (Exception e) {}
          }
     }
     // головний потік запускає три потоку
     public static void main (String [] argv) {
          new test("A", 500); //перший
          new test("B", 700); //другий
          new test("C", 1000); // третій
     }
}
```

17.16 Єдиність способу запуску потоку.

Запустити дочірній потік можна тільки за допомогою методу start.

Якщо на об'єкті потоку викликати метод run, то це призведе лише до одноразового виконання тіла даного методу в тому потоці, який викликав run.

```
public class test extends Thread {
     public void run() {
          Thread t = Thread.currentThread();
          System.out.println (t.getName());
     }
     public static void main (String [] argv) {
          test t = new test();
          t.run();       // main
          t.start();     // thread-0
     }
}
```

Зауваження. JVM задає кожному потоку деякий системне ім'я при його створенні

17.17 Ім'я потоку

При створенні об'єкта потоку JVM присвоїть потоку деякий ім'я. Ім'я можна отримати за допомогою методу getName, якщо його викликати на відповідному об'єкті-потоці.

```
t.getName();
```

Якщо потрібно отримати ім'я потоку, який виконує певний метод, досить в код методу вставити наступну інструкцію:

```
String threadName = Thread.currentThread().getName();
```

18 УПРАВЛІННЯ ВЗАЄМОДІЄЮ ПОТОКІВ

18.1 Паралельне виконання одного коду різними потоками

Два різних потоку можуть виконувати один і той же код, це може привести до небажаних наслідків.

```
public class Test {
     private boolean flag;
      // виконують два потоку паралельно
     public void m (boolean flag) {
          this.flag = flag;
          try {Thread.sleep (200);} // чекаємо
          catch (InterruptedException e) {
               e.printStackTrace();
          }
          System.out.println (this.flag + "==" + flag);
     } // true == false

     public static void main (String [] args) {
          new Thread() {// створення першого потоку
               public void run() {
                    while (true) m (true);
               }
          } .start(); // запуск першого потоку
          new Thread() {// створення другого потоку
               public void run() {
                    while (true) m (false);
               }
          } .start(); //запуск другого потоку
     }
}
```

18.2 Синхронізація

Для вирішення проблеми паралельного доступу різних потоків до одного і того ж коду пріменяютт синхронізацію. Синхронізації може бути підданий метод, статичний метод, блок коду.

метод	
статичний метод	public synchronized void m() {...}
блок коду	public static synchronized void m() {...}
ділянка коду	synchronized (M) {...}

Синхронізація здійснюється за допомогою спеціального об'єкта-монітора, який пов'язаний з цією синхронізацією. Коли потік починає виконувати синхронізований код, кажуть, що він блокує відповідний монітор (або володіє монітором). Якщо потоку потрібно виконати синхронізований код, а монітор заблокований іншим потоком, то він знаходиться в блокованому стані, поки з монітора не буде знято блокування.

18.3 монітор синхронізації

Монітором завжди є об'єкт, який залежить від того, який код синхронізується.

Зауваження. Монітором не може бути null.

Зауваження. Оскільки JVM створює унікальний об'єкт типу Class, для кожного завантаженого класу, то для статичних синхронізованих методів існує один і тільки один монітор.

Для нестатичних синхронізованого методу може існувати кілька моніторів - по числу створених об'єктів класу, в якому визначено цей метод.

```
public class Test extends Thread {
     public synchronized void m() {
          while (true) {
               System.out.println (
                    Thread.currentThread().getName()));
          }
     }
     public Test() {start();}
     public void run() {m();}
     public static void main (String [] argv) {
          Test treadA = new Test();
          Test threadB = new Test();
     }
     // в процесі виконання будуть виводитися
     // імена потоків threadA і threadB упереміш
}
```

В даному прикладі два потоку A і B виконують один і той же метод одночасно, тому що вони блокують різні монітори: threadA і threadB відповідно, в такому випадку синхронізації немає).

Якщо метод m зробити статичним, то на екран повідомлення буде виводити тільки один потік A. Другий потік B буде нескінченно довго чекати, поки перший не розблокує монітор, який в даному випадку один: об'єкт класу Class, відповідний класу Test (Test.class).

18.4 Інструкція synchronized

Інструкція synchronized дозволяє синхронізувати довільний ділянку код.

```
public class Test {
     // об'єкт-монітор
     private static final Object MONITOR = new Object();
     public void m() {
          synchronized (MONITOR) {...}
     }
}
```

Зауваження. Параметром інструкції synchronized завжди є об'єкт.

18.5 Метод join класу Thread

Клас Thread містить метод join, призначений для перекладу потоку, який викликав цей метод в режим паузи до тих пір, поки не закінчить своє виконання потік, на якому цей метод був викликаний.

Зауваження. Метод викидає виключення InterruptedException, якщо в потоці, який його викликав встановити т.зв. внутрішній прапор переривання потоку (викликати на цьому ж об'єкті потоку метод interrupt).

Зауваження. Потік, що виконує метод join не знімає блокування з моніторів, які він заблокував.

```
public class Test extends Thread {
     public void run() {
          try {sleep (2000); }
          catch (Exception ex) {ex.printStackTrace(); }
     }

     public static void main (String [] argv)
               throws Exception {
          Test t = new Test();
          t.start();
          t.join();
          System.out.println("Pause has been expired ");
     }
}
```

18.6 Метод wait класу Object

Метод wait переводить потік в режим очікування. Метод має три перевантажених варіанту.

Метод wait повинен викликатися

- тільки з синхронізованого коду;
- на об'єкті моніторі, пов'язаних з даними синхронізованим кодом;
- потік, який викликає ці методи, повинен володіти монітором.

Метод wait може викинути виняток InterruptedException.

Зауваження. При виклику методу wait на моніторі, блокування з цього монітора знімається і потік переходить в режим очікування на цьому моніторі.

Зауваження. Метод wait визначено в класі Object. Монітором може бути будь-який об'єкт.

```
public class Test extends Thread {
     public void run() {
          synchronized (this) {
               try {wait();}
               catch (Exception ex) {
                    System.out.println (
                         "Thread has been interrupted");
               }
          }
     }
```

```
    public static void main (String [] args)
              throws Exception {
         Test t = new Test();
         t.start();

         // пауза в головному потоці, щоб встиг
         // почати своє виконання метод wait:
         Thread.sleep (500);

         // перервати запущений потік:
         t.interrupt();
    }
}
```

Зауваження. Коли потік виконує метод wait можливі т.зв. випадкові пробудження. Тому виконання методу wait слід укладати в цикл з перевіркою деякого умови. Контракт класу Object:

```
synchronized (obj) {
while (<condition does not hold>)
    obj.wait (timeout, nanos);
    ... // Perform action appropriate to condition
}
```

18.7 Методи notify і notifyAll класу Object

Для виходу потоку з режиму очікування застосовують методи notify і notifyAll, які повинні:

- викликатися на об'єкті-моніторі
- тільки з синхронізованого коду.
- потік, який викликає ці методи, повинен володіти монітором.

На одному і тому ж моніторі може перебувати кілька потоків в режимі очікування. Метод notify пробуджує тільки один випадково обраний потік; notifyAll пробуджує все потоки, які очікують на цьому моніторі.

Зауваження. Методи notify і notifyAll визначені в класі Object, тому що монітором потенційно може бути будь-який об'єкт.

Зауваження. Методи notify і notifyAll викидають IllegalMonitorStateException з повідомленням «current thread not owner», якщо потік, який викликає ці методи, не є власником монітора.

Зауваження. Небажано використовувати метод notify, тому що невідомо який саме потік отримає повідомлення.

18.8 Блоковане стан - пробудженого потоку

Якщо потік перебував в стані очікування, виконуючи метод wait і був пробуджений методом notify / notifyAll то він знаходитися в блокованому стані до тих пір, поки не звільнитися монітор і тільки після цього продовжує виконувати першу інструкцію за методом wait.

```
public class Test extends Thread {
     public void run() {
          synchronized("abc") {
               try {
                    "Abc" .wait();
                    System.out.println (
                         "Thread has been notified");
               }
               catch (Exception e) {e.printStackTrace(); }
          }
     }

     public static void main (String [] argv)
                    throws Exception {
          new Test().start();
          Thread.sleep (500);
          synchronized("abc") {
               "Abc" .notifyAll();
               Thread.sleep (3000);
          }
     }
}
```

18.9 Спільне використання методів wait і notify / notifyAll

Спільне використання методів notify / notifyAll і wait дає можливість потокам синхронізувати своє виконання.

Зауваження. Якщо на моніторі немає очікують потоків, то виклик методів notify / notifyAll не генерує ніяких виняткових ситуацій.

```
public class Test extends Thread {
     private static final Object LOCK = new Object();
     private String mes;
     public Test (String mes) {this.mes = mes;}
     public void run() {
          synchronized (LOCK) {
               try {
                    while (true) {
                         LOCK.wait();
                         LOCK.notify();
                         System.out.println (mes);
                    }
               }
               catch (Exception e) {e.printStackTrace();}
          }
     }

     public static void main (String [] argv)
                    throws Exception {
          // після запуску обидва потоку переходять в
          // режим очікування, виконуючи wait
          new Test("A").start();
```

```
          new Test("B").start();
          // пауза, щоб запущені потоки встигли
          // почати виконять wait
          Thread.sleep (500);
          synchronized (LOCK) {
               // для оповіщення всіх чекають на
               // моніторі потоків потрібно викликати на
               // ньому метод notifyAll
               LOCK.notifyAll();
          }
     }
}
```

Потоки можуть синхронізувати своє виконання без участі третього (головного) потоку.

```
public class Test extends Thread {
     private String mes;
     public Test (String mes) {this.mes = mes;}

     public static void main (String [] argv)
                    throws Exception {
          new Test("A").start();
          new Test("B").start();
     }

public void run() {
          synchronized (Test.class) {
               try {
                    while (true) {
                         Test.class.notify();
                         Test.class.wait();
                         System.out.println (mes);
                    }
               }
               catch (Exception e) {e.printStackTrace();}
          }
     }
}
```

18.10 взаємне блокування

Потоки можуть входити в стан взаємного блокування.

```
public class Test extends Thread {
     private static final Object LOCK_A = new Object();
     private static final Object LOCK_B = new Object();

     public void run() {
          try {
               synchronized (LOCK_A) {
                    // після паузи потік чекає, поки
                    // буде знято блокування з
```

```
                    // монітора LOCK_B
                    sleep (500);
                    synchronized (LOCK_B) {}
               }
          } Catch (Exception e) {e.printStackTrace();}
     }

     public static void main (String [] args)
                    throws Exception {
          new Test().start();
          synchronized (LOCK_B) {
               // після паузи потік чекає, поки буде
               // знято блокування з монітора
               // LOCK_A
               sleep (300);
               synchronized (LOCK_A) {}
          }
     }
}
```

18.11 стану потоків

Клас Thread містить вкладений enum State, елементи якого являють унікальні стану потоку. Стан потоку повертає Thread # getState.

NEW	"новий"	Потік створений, але ще не запущений.
RUNNABLE	"Працездатний"	Потік виконується в JVM.
BLOCKED	"Блокований"	Потік заблокований на моніторі.
TERMINATED	"Зупинений"	Потік завершив виконання.
WAITING	"Очікує"	Потік виконує wait / join (без параметрів).
TIMED_WAITING	"Очікує встановлений час"	Потік виконує wait / join / sleep (с параметрами)

18.12 Метод interrupt класу Thread

Якщо потік знаходиться в стані WAITING / TIMED_WAITING виконуючи методи sleep / join / wait, а інший потік викликає на цьому потоці метод interrupt, то відповідні методи припиняють своє виконання і викидають виняток InterruptedException.

Зауваження. Внутрішній прапор переривання потоку в даному випадку встановлено не буде.

```
public class Test extends Thread {
     public void run() {
          try {
               sleep (10000); // InterruptedException
```

```
                join(); // InterruptedException
                synchronized (Test.class) {
                     // InterruptedException:
                     Test.class.wait();
                }
           } Catch (Exception e) {
                e.printStackTrace();
           }
     }

     public static void main (String [] argv)
                throws Exception {
          Test t = new Test();
          t.start();
          t.interrupt();
     }
}
```

18.13 Методи interrupt / isInterrupted

Метод interrupted перевіряє чи був поточний потік перерваний і скидає внутрішній прапор в false.

```
public static boolean interrupted()
```

Метод isInterrupted перевіряє чи був потік (на якому даний метод викликаний) перерваний, внутрішній прапор при цьому залишається без змін.

```
public boolean isInterrupted()
```

перевіряє чи був потік (на якому даний метод викликаний) перерваний, внутрішній прапор при цьому залишається без змін.

Приклад:

```
Thread t = Thread.currentThread();
t.interrupt();

// true
System.out.println (t.isInterrupted());

// true
System.out.println (t.isInterrupted());

// true
System.out.println (Thread.interrupted());

// false
System.out.println (Thread.interrupted());
```

18.14 Зміна значення аргументу - блоку синхронізації

Присвоєння аргументу блоку синхронізації коду нового значення не призводить до зміни монітора в ньому. Якщо в якості аргументу

синхронізованих блоків передбачається використовувати спеціально створене для цих цілей поле, то доцільно оголошувати його з модифікатором final.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. The Java Language Specification. Java SE 8 Edition. [Електронний ресурс] http://docs.oracle.com/javase/specs/jls/se8/jls8.pdf

Електронне Навчальне видання

КОНСПЕКТ ЛЕКЦІЙ
з дисципліни
"Розробка корпоративних застосування"

для студентів усіх форм навчання
спеціальності
7.05010302 - "інженерія програмного забезпечення"

Упорядник: Колесников Дмитро Олегович.

Відповідальний за випуск
редактор
Комп'ютерна верстка

Авторська редакція

План 2017 р. (), Поз. 00.

Підп. до друку 00.00.2017 р. формат 60×841/16. Способ друку - різографія.
Умів. друк. арк. 0,0. Облік. вид. арк. 0,0. Тираж 00 прим.
Зам. № 0-00. Ціна договірна.

ХНУРЕ, Україна, 61166, м. Харків, просп. Науки, 14

Надруковано в навчально-науковому видавничо-поліграфічному центрі
ХНУРЕ,
Україна, 61166, м. Харків, просп. Леніна, 14.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

МЕТОДИЧНІ ВКАЗІВКИ
до самостійної роботи з дисципліни
"ОСНОВИ ПРОГРАМУВАННЯ НА JAVA"

для студентів усіх форм навчання
напряму підготовки
6.050103 – "Програмна інженерія"

Електронне видання

Харків 2017

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

МЕТОДИЧНІ ВКАЗІВКИ
до самостійної роботи з дисципліни
"ОСНОВИ ПРОГРАМУВАННЯ НА JAVA"

для студентів усіх форм навчання
напряму підготовки
6.050103 – "Програмна інженерія"

Електронне видання

ЗАТВЕРДЖЕНО
кафедрою ПІ.
Протокол № ___ від _____ р.

Харків 2017

Методичні вказівки до самостійної роботи з дисципліни "Основи програмування на Java" для студентів усіх форм навчання першого (бакалаврського) рівня вищої освіти напряму підготовки 6.050103 – "Програмна інженерія". [Електронне видання] / Упоряд.: Д.О. Колесников. – Харків: ХНУРЕ, 2017. – 20 с.

Упорядник: А.В. Колесников, доцент кафедри ПІ.

Рецензент:

ЗМІСТ

ВСТУП

Метою самостійної роботи студентів з дисципліни "Основи програмування на Java" є самостійне закріплення теоретичного, практичного та лабораторного матеріалів.

Під час опрацювання теоретичного матеріалу студенти користуються загальною та практичною інформацією, яка наведена в даних методичних вказівках та яка рекомендована у спеціальній літературі.

Студенти закріплюють опрацьований матеріал відповідями на запитання для самоконтролю, які наведені в методичних вказівках. Самостійна робота над дисципліною має бути систематичною протягом всього періоду вивчення дисципліни.

Лекції призначені для вивчення головних теоретичних положень курсу. Обсяг лекцій не дозволяє вивчити всі розділи програми, тому частину теоретичних розділів студенти повинні вивчати самостійно. Для найкращого освоєння теоретичного матеріалу рекомендується перед кожною лекцією прочитати матеріал попередньої лекції, а також рекомендований матеріал для самостійного вивчення.

Практичні заняття мають дуже важливе значення, тому що на них студенти починають досконаліше розуміти теоретичний матеріал. До практичного заняття треба готуватися по методичнім вказівкам до практичних занять.

Мета лабораторної роботи – закріпити теоретичні та практичні знання за допомогою вирішення задач відповідно до завдання із урахуванням новітніх технологій. Кожна лабораторна робота захищається з відміткою із урахування шкали оцінювання. Лабораторна робота виконується та захищається індивідуально.

Кожна лабораторна робота захищається кожним студентом індивідуально.

1 Мета і задачі дисципліни

1.1 Мета дисципліни

– *Метою викладання навчальної дисципліни є*: формування у студентів теоретичних знань та практичних навичок об'єктно-орієнтованого програмування з використанням платформи Java Standard Edition.

1.2 Задачі дисципліни

За результатами вивчення дисципліни студенти мають вирішити такі задачі:

– вивчити мову програмування Java;

– ознайомитися з класами ядра Java;

– ознайомитися з принципами об'єктно-орієнтованої розробки застосувань мовою Java.

1.3 Програма знань і умінь

За результатами вивчення дисципліни студенти повинні знати:

- мову програмування Java;
- основні класи ядра платформи JSE;
- принципи об'єктно-орієнтованої розробки застосувань мовою Java.

За результатами вивчення дисципліни студенти повинні вміти:

- користуватися принципами об'єктно-орієнтованої розробки для написання застосувань мовою Java за допомогою платформи JSE;
- використовувати інтегровані середовища для розробки застосувань на Java.

2 Робоча програма дисципліни

2.1 Теми дисципліни

Тема 1. Введення в платформу Java. Кодування.

Тема 2. Лексична трансляція. Типи даних та літерали.

Тема 3. Операції та оператори.

Тема 4. Класи і конструктори. Перетворення між типами.

Тема 5. Абстрактні класи та інтерфейси. Вкладені типи.

Тема 6. Перелічувальний тип.

Тема 7. Generics. Параметризовані класи та методи.

Тема 8. Строкові класи. Інтернаціоналізація.

Тема 9. Регулярні вирази.

Тема 10. Механізм винятків.

Тема 11. Потоки введення / виводу.

Тема 12. Порівняння об'єктів.

Тема 13. Колекції.

Тема 14. Асоціативні контейнери.

Тема 15. Багатопоточне програмування.

2.2 Розділи робочої програми для самостійного вивчення

З програми курсу видно, що на кожну тему відводиться недостатньо часу. На лекції даються основні поняття, показується практична значущість розділу, розглядаються найбільш складні питання. Студент повинен самостійно розібратися в додаткових питаннях по кожному розділу. Для цього потрібно вивчити розділ по підручниках і додатковій літературі, яка розглядається в методичних вказівках. На лекціях розглядаються обов'язкові розділи, які потрібно додатково розглянути по підручниках. Для більш глибокого розуміння даного курсу і отримання більш повних знань потрібно деякі теми з підручників розглянути самостійно. Нижче наведено список цих розділів.

№	Назва теми	Кількість годин	
		денна	заочна
1	Введення в платформу Java. Кодування.	6	8
2	Лексична трансляція. Типи даних та літерали.	6	8
3	Операції та оператори.	6	8
4	Класи і конструктори. Перетворення між типами.	6	8
5	Абстрактні класи та інтерфейси. Вкладені типи.	6	8
6	Перелічувальний тип.	5	8
7	Строкові класи. Інтернаціоналізація.	5	8
8	Регулярні вирази.	5	8
9	Generics. Параметризовані класи та методи.	5	8
10	Механізм винятків.	5	7
11	Потоки введення / виводу.	5	7
12	Порівняння об'єктів.	5	7
13	Колекції.	5	7
14	Асоціативні контейнери.	5	7
15	Багатопоточне програмування.	5	7
	Загальна кількість	80	114

2.3 Рейтингове оцінювання успішності студентів з дисципліни

Формою підсумкового контролю з дисципліни "Основи програмування на Java" є залік.

Усне опитування студентів допомагає контролювати не лише знання, а й вербальні вміння, сприяє виправленню мовних помилок. Відтворення студентом раніше вивченого матеріалу сприяє кращому запам'ятовуванню, активному використанню наукових понять, що неможливо без достатнього застосування їх у мові.

Усне опитування може бути індивідуальним і фронтальним. За фронтального опитування студенти відповідають з місця, доповнюючи один одного. Частковим випадком фронтального опитування є групове опитування – 5-6 осіб одночасно. Індивідуальне опитування здійснюється у процесі проведення співбесіди під час практичних занять.

Запитання для усної перевірки знань поділяють на основні, додаткові і допоміжні. Основні запитання передбачають самостійну розгорнуту відповідь (наприклад, запитання щодо змісту лабораторного заняття). Додаткові спрямовані на уточнення того, як студент розуміє певне питання, формулювання, формулу та ін. Допоміжні запитання мають за мету виправлення помилок та неточностей, якщо такі мали місце у відповіді студента. Усі запитання повинні бути логічними, чіткими, зрозумілими, а їх сукупність – послідовна і системна.

Письмовий контроль можна здійснюється у вигляді відповідей на запитання, розв'язання задач під час виконання практичних робіт. Письмові роботи допомагають за короткий час з'ясувати рівень засвоєння матеріалу у великої кількості студентів. Результати письмових робіт можна проаналізувати і з'ясувати деталі і неточності у відповідях та аналізувати їх причини.

Формою письмового контролю з дисципліни "Основи програмування на Java" є комплексна контрольна робота (КР), яка проводиться на останньому практичному занятті (ПЗ).

Практичний контроль передбачає виявлення вмінь і навичок студентів, що набуті під час практичної діяльності (практичні заняття, робота над власним проектом). Така перевірка дає змогу виявити, на якому рівні студент засвоїв теоретичні основи цих дій.

Як форма підсумкового контролю для дисципліни використовується залік.

2.4 Розподіл балів.

Кількісні критерії оцінювання:

Вид заняття/ контрольний захід	Оцінка Осем	
	Денна	Заочна
Лб № 1	6-10	12-20
Лб № 2	6-10	12-20
Лб № 3	6-10	12-20
Лб № 4	6-10	
Лб № 5	6-10	
Пз № 1	6-10	12-20
Пз № 2	6-10	12-20
Пз № 3	6-10	
Пз № 4	6-10	
Пз № 5	6-10	
Всього за семестр	60-100	60-100

2.5 Якісні критерії оцінювання.

2.5.1 Необхідний обсяг знань для одержання позитивної оцінки

− знати мову програмування Java;

− знати основні класи ядра платформи JSE;

− знати принципи об'єктно-орієнтованої розробки застосувань мовою Java.

2.5.2 Необхідний обсяг умінь для одержання позитивної оцінки

− вміти користуватися принципами об'єктно-орієнтованої розробки для написання застосувань мовою Java за допомогою платформи JSE;

− використовувати інтегровані середовища для розробки застосувань на Java.

2.5.3 Критерії оцінювання роботи студента протягом семестру

Як форма підсумкового контролю для дисципліни використовується залік. При цьому виді контролю підсумкова оцінка $Р_п$ обчислюється за формулою $Р_п = О_{сем}$, де – $О_{сем}$ оцінка за семестр в 100 бальній системі.

Критерії оцінювання роботи студента протягом семестру.

Задовільно D, E (60-74). Мати мінімум знань та вмінь. Виконати та захистити усі лабораторні роботи. Виконати всі завдання під час практичних занять.

Добре C (75-89). Твердо засвоїти мінімум знань. Вміти використовувати ці знання при розв’язанні практичних завдань. Виконати та захистити усі лабораторні роботи в строк. Виконати усі пункти завдань до практичних занять в строк.

Відмінно A, B (90-100). Знати усі теми та вільно орієнтуватися у предметній галузі дисципліни. Виконати та захистити усі лабораторні роботи та практичні завдання в строк з найвищою оцінкою. Виконати усі індивідуальні завдання до практичних занять в строк з найвищою оцінкою.

2.5.4 Критерії оцінювання знань та вмінь студента під час заліку

Задовільно D, E (60-74). Показати необхідний мінімум теоретичних знань. Вміти використовувати ці знання при розв’язанні практичних задач.

Добре C (75-89). Твердо орієнтуватися в теоретичній складовій курсу. Вміти створювати на практиці застосування мовою Java. Використовувати інтегровані середовища для розробки застосувань на Java..

Відмінно A, B (90-100). Показати максимально повні знання в предметній галузі дисципліни. Обґрунтувати рішення практичної задачі.

2.5.5 Шкала оцінювання: національна та ЄКТС

Сума балів за всі види навчальної діяльності	Оцінка ЄКТС	Оцінка за національною шкалою	
		для іспиту, курсового проекту (роботи), практики	для заліку
96-100	A	відмінно	зараховано
90-95	B		
75-89	C	добре	
66-74	D	задовільно	
60-65	E		
35-59	FX	незадовільно з можливістю повторного складання	не зараховано з можливістю повторного складання
0-34	F	незадовільно з обов'язковим повторним вивченням дисципліни	не зараховано з обов'язковим повторним вивченням дисципліни

2.6 Рекомендована література

2.6.1 Базова література

1. Шилдт Г. Java 8. Полное руководство : пер. с англ. / Г. Шилдт. – 9-е изд. – М.: Вильямс, 2015. – 1376 с.

2. The Java Language Specification. Java SE 8 Edition. [Електронний ресурс] http://docs.oracle.com/javase/specs/jls/se8/jls8.pdf

3. Конспект лекцій з дисципліни "Основи програмування на Java" для студентів усіх форм навчання напряму підготовки 6.050103 – "Програмна інженерія" [Електронний ресурс] / ХНУРЕ; Д.О. Колесников – Харків: ХНУРЕ, 2017. – 103 с. Електронна версія (pdf / 742 Kb).

4. Методичні вказівки до практичних робіт з дисципліни "Основи програмування на Java" для студентів усіх форм навчання напряму підготовки 6.050103 – "Програмна інженерія" [Електронний ресурс] / ХНУРЕ; розроб. Д.О. Колесников. – Харків: ХНУРЕ, 2017. – 8 с. Електронна версія (pdf / 371 Kb).

5. Методичні вказівки до лабораторних робіт з дисципліни "Основи програмування на Java" для студентів усіх форм навчання напряму підготовки 6.050103 – "Програмна інженерія" [Електронний ресурс] / ХНУРЕ; розроб. Д.О. Колесников. – Харків: ХНУРЕ, 2017. – 20 с. Електронна версія (pdf / 513 Kb).

2.6.2 Допоміжна література

6. The Java Tutorials. [Електронний ресурс] http://docs.oracle.com/javase/tutorial

7. Хорстманн К., Корнелл Г. Java 2. Библиотека профессионала. Том первый. Основы. - М.: Вильямс, 2014. - 1008 с.

8. Хорстманн К., Корнелл Г. Java 2. Библиотека профессионала. Том второй. Тонкости программирования. - М.: Вильямс, 2014. - 864 с.

2.6.3 Інформаційні ресурси

9. http://www.oracle.com/technetwork/java/javase/tech/index.html

3 Методичні вказівки з вивчення дисципліни

3.1 Методи навчання

Методами навчання дисципліни "Основи програмування на Java" є:

1. Методи організації та здійснення навчально-пізнавальної діяльності за допомогою слайд-лекцій, пояснень на багатьох практичних прикладів.

2. Методи стимулювання й мотивації навчально-пізнавальної діяльності студентів у виконанні власних проектів з практичної реалізації завдань дисципліни.

3. Методи контролю (самоконтролю, корекції) за ефективністю навчально-пізнавальної діяльності студента – це є методи, які спрямовані на самостійну, творчу та пізнавальну діяльність студентів, особливо при створені власних проектів.

4. Універсальні методи поєднують самостійну роботу студентів під час практичних занять з інструктуванням, допомогою викладача, у результаті чого студенти набувають навичок щодо проведення самостійної роботи поза аудиторного навантаження. Крім цього студент має у своєму розпорядженні слайд-лекції, приклади розв’язання задач з роз’ясненнями, які поєднуються в наочно-ілюстративно-практичний комплект матеріалів для навчання.

3.2 Теоретична частина

Рекомендуємо наступний порядок вивчення теоретичних розділів курсу:

1. Теоретичні розділи курсу вивчати в порядку, в якому вони наведені в програмі. Не рекомендується пропускати окремі розділи. Нижче наведено розгорнутий план лекцій з дисципліни.

2. Обов’язково відповідати на запитання, які наведені в кінці розділу.

3.3 Розгорнутий план лекцій

Метою лекцій є формування у студентів системи знань у галузі розробки застосувань мовою Java.

Лекція № 1: Введення в платформу Java. Кодування.

1.1. Введення в платформу.

1.1.1. JSE / JEE / JME, JVM, JRE, JDK, мова Java.

1.1.2. Специфікація мови (документ).

1.1.3. Code convention.
1.1.4. Компіляція і запуск з консолі.
1.1.5. Classpath.
1.1.6. Eclipse IDE.

1.2. Кодування.
1.2.1. Кодування ASCII, керуючі символи CR, LF.
1.2.2. Приклад застосування керуючих символів (в консолі).
1.2.3. Unicode, кодова точка Unicode.
1.2.4. Нотація позначення символів Unicode.
1.2.5. Діапазони символів Unicode (ASCII, ISO-8859-1, BMP).
1.2.6. UTF і їх види.
1.2.7. Мітка порядку байт BOM, і її використання.
1.2.8. Подання додаткових символів сурогатними (UTF-16BE).
1.2.9. Unicode escape послідовності.
1.2.10. Приклад (висновок скрипкового ключа в консоль).
1.2.11. Кодування вихідного коду програми.
1.2.12. Кодування за замовчуванням OS.
1.2.13. Однобайтні кодування Cp1251, Cp866, ANSI, ISO-8859-1.

Лекція № 2: Лексична трансляція. Типи даних та літерали.

2.1. Лексична трансляція.
2.1.1. Обмежувачі рядків.
2.1.2. Роздільники лексем.
2.1.3. Пробільні символи.
2.1.4. Коментарі.
2.1.5. Лексеми мови.

2.2. Ідентифікатори.
2.2.1. Структура ідентифікаторів.
2.2.2. Буква, буква або цифра Java.
2.2.3. Ключові слова:
(1) примітивні типи даних;
(2) модифікатори рівня доступу;
(3) використовувані в операторах вибору;
(4) використовувані в циклах;
(5) використовувані при роботі з винятками;
(6) невикористані.

2.3. Типи даних.
2.3.1. Довідкові.
2.3.2. Примітивні.

2.4. Числові літерали.
2.4.1. Числовий літерал зі знаком.
2.4.2. Цілі літерали і їх тип.
2.4.3. Подання негативних чисел цілими літералами.
2.4.4. Діапазони десяткових цілих литералов.

2.4.5. Структура цілих литералов:
(1) шістнадцятирічних;
(2) десяткових;
(3) вісімкових;
(4) бінарних.
2.5. Речові літерали.
2.5.1. Тип речових літералів.
2.5.2. Структура речових літералів:
(1) десяткових;
(2) шістнадцятирічних.
2.5.3. Структура експоненти:
(1) десяткової;
(2) шестнадцатеричной.
2.6. Булеві літерали.
2.7. Символьні літерали.
2.8. Рядкові літерали.
2.8.1. Способи подання.
2.8.2. Конкатенація строкових літералів.
2.8.3. Escape послідовності:
(1) символьні;
(2) восьмеричні.
2.9. Роздільники.
2.10. Операції (оглядово).

Лекція № 3: Операції та оператори. Масиви.
3.1. Перетворення між примітивними типами:
3.1.1. розширює;
3.1.2. звужує (+ з втратою інформації).
3.2. Оптимізація компілятора при обчисленні констант.
3.3. final локальні змінні.
3.4. Операції і оператори.
3.4.1. Арифметичні операції.
3.4.2. Операції інкремента, декремента, префиксная, постфіксний форми.
3.4.3. Оператор багатозначного вибору, тип селектора.
3.4.4. Тернарний оператор.
3.4.5. Оператори циклу.
3.4.6. Оператори управління.
3.4.7. Операції з привласненням.
3.4.8. Оператори зсуву.
3.4.9. Логічні та побітові операції.
3.4.10. Логічні операції по короткій схемі.
3.4.11. Пріоритет і асоціативність операцій і операторів.
3.5. Тип арифметичного виразу.

3.6. Масиви.
3.6.1. Одномірні.
3.6.2. Багатовимірні (індуктивне визначення).
3.6.3. Анонімні.
3.6.4. Константи.
3.6.5. Varargs.

Лекція № 4: Класи і конструктори. Перетворення між типами.
4.1. Класи.
4.1.1. ООП в Java, класи і об'єкти.
4.1.2. Оператор new.
4.1.3. Вміст класу:
(1) конструктори;
(2) ключове слово static;
(3) блоки ініціалізації (static / not static);
(4) методи (static / not static);
(5) поля (static / not static);
(6) вкладені типи (static / not static).
4.1.4. Значення полів за замовчуванням.
4.1.5. Ініціалізація полів.
4.1.6. Спадкування, інкапсуляція, поліморфізм.
4.1.7. Пакети (іменування, приклади з ядра, використання).
4.1.8. Клас Object (основні методи).
4.1.9. Рівні доступу елементів класу, самого класу.
4.1.10. Оператор instanceof.
4.1.11. Перетворення між посилальними типами:
(1) між класами;
(2) між масивами.
4.1.12. Перевантаження, перекриття та приховування методів.
4.1.13. Обмеження при перекритті і приховуванні методів.
4.1.14. Ключове слово final для сутностей:
(1) клас;
(2) поле;
(3) метод, статичний метод;
(4) локальна змінна;
(5) параметр методу.
4.1.15. Ініціалізація статичних і не статичних (final) полів.
4.1.16. Клас Class (призначення, способи отримання).
4.1.17. GC.
4.2. Конструктори.
4.2.1. Конструктори, їх відміна від методів.
4.2.2. Конструктор за замовчуванням.
4.2.3. Рівні доступу конструкторів.
4.2.4. Ключові слова this (3 контексту), super (2 контексту).

4.2.5. Блоки ініціалізації (static / not static).

4.2.6. Порядок виклику конструкторів / блоків ініціалізації при створенні.

4.2.7. Об'єкта.

Лекція № 5: Абстрактні класи та інтерфейси. Вкладені типи.

5.1. Вкладені класи.

(1) Види класів по оголошенню (class / enum);

(2) Види класів по місту розташування декларації:

(a) вкладені;

(b) внутрішні;

(c) локальні;

(d) анонімні;

(e) елементи інших довідкових типів.

5.2. Абстрактні класи.

5.2.1. Абстрактні класи, їх властивості та призначення.

5.2.2. Конструктори абстрактних класів.

5.2.3. Абстрактні методи, обмеження на модифікатори і специфікатор.

5.2.4. Спадкування абстрактного класу.

5.3. Інтерфейси.

5.3.1. Інтерфейси і їх вміст.

5.3.2. Відмінність абстрактних класів від інтерфейсів.

5.3.3. Специфікатори елементів інтерф. за замовчуванням:

(1) полів;

(2) методів;

(3) вкладених типів.

5.3.4. Методи за замовчуванням в інтерфейсах (Java8 +).

5.4. Оболонки.

5.4.1. Класи оболонки. Boxing, unboxing.

5.4.2. Кешування числових оболонок.

Лекція № 6: Перелічувальний тип.

6.1. Призначення, оголошення.

6.2. Основні методи класу Enum.

6.3. Структура реализации перечислимого типу JVM.

6.4. Приклад класу відповідного перераховуваті типу.

6.5. Обмеження на использование.

Лекція № 7: Generics. Параметризовані класи та методи.

7.1. Узагальнення, призначення.

7.2. Generic-типи, параметризація, сирої тип.

7.3. Wildcard параметризрвані типи:

7.3.1. extends wildcard;

7.3.2. super wildcard;
7.3.3. unbounded wildcard.
7.4. Параметри Generic-типів з обмеженнями.
7.5. Обмеження використання параметра Generic-типів.
7.6. Перетворення між параметризрвані типами.
7.7. Generic-методи.
7.7.1. Приклади з ядра.
7.7.2. Параметризація Generic-методів.
Лекція № 8: Строкові класи. Інтернаціоналізація.
8.1. Строки.
8.1.1. Клас String.
(1) Конструктори, конструктор копіювання.
(2) Основні методи.
(3) Порівняння рядків.
(4) Locale-sensitive (Collator).
8.1.2. Операція + для рядків.
8.1.3. Змінні рядка StringBuilder / Buffer.
8.1.4. Приклад, з'єднання рядків String vs StringBuilder / Buffer:
(1) швидкість роботи;
(2) споживана пам'ять (дивимося профайлером);
8.1.5. Рядкові літерали.
8.1.6. Пул рядків, приклад.
8.2. Інтернаціоналізація.
8.2.1. Інтернаціоналізація, локалізація, відмінності.
8.2.2. Класи Locale, ResourceBundle.
8.2.3. Локаль за замовчуванням, коди країн, мов.
8.2.4. Пакети ресурсів (classes, properties), їх імена.
8.2.5. Алгоритм завантаження пакета ресурсів.
8.2.6. Алгоритм пошуку значення по ключу.
8.2.7. Приклади.
Лекція № 9: Регулярні вирази.
9.1. Визначення, для чого потрібні.
9.2. Символьні класи.
9.3. Символьні класи Java.
9.4. Зумовлені класи.
9.5. Межі.
9.6. Обмежувачі рядків.
9.7. Квантіфікатори (жадібні, ледачі).
9.8. Сверхжадние квантіфікатори.
9.9. Логічні операції.
9.10. Групи (в т.ч. застосування в самому РВ).
9.11. Екранування символів.
9.12. Перегляд вперед / назад позитивний / негативний.
9.13. Режими, як включити.

9.14. Класи Matcher, Pattern.
9.15. Приклад на Java.
9.16. Порівняння РВ Java з РВ на інших платформах.

Лекція № 10: Механізм винятків.
10.1. Визначення.
10.2. Клас Throwable, його вміст (getMessage / toString, конструктори).
10.3. Основні класи в ієрархії.
10.4. Перевіряються і непроверяемие виключення.
10.5. Реакція на виключення (обробка / випуск назовні).
10.6. Стек викликів.
10.7. Блок try / catch / finally.
10.8. Ключове слово throws.
10.9. Оператор throw.
10.10. Множинний catch (multi-catch).
10.11. Вихід з return в catch / finally, приклад.
10.12. Try-with-resources, Autocloseable.
10.13. Пригнічені виключення, їх отримання, приклад.
10.14. Викид виключення в catch.
10.15. Обмеження при перекритті / приховуванні методу.
10.16. Обхід вищевказаного обмеження (приклад).
10.17. Власні класи виняток.
10.18. Ключове слово assert.
10.19. Як використовувати (чи використовувати взагалі), приклад (-ea опція).

Лекція № 11: Потоки введення / виводу.
11.1. Потоки введення / виведення ядра.
11.1.1. Байтові і символьні потоки.
11.1.2. Ієрархії InputStream / OutputStream, Reader / Writer.
11.1.3. Основні класи.
11.1.4. Парні потоки.
11.1.5. Буферизація.
11.1.6. Клас RandomAccessFile.
11.1.7. Мости InputStreamReader / OutputStreamWriter.
11.1.8. Перекодування.
11.1.9. Кодування за замовчуванням.
11.1.10. Приклад низкоуровневого введення (System.in).
11.2. Клас File.
11.2.1. Елементи файлової системи.
11.2.2. Посилання (hard / soft link, junction point).
11.2.3. Відносний шлях і абсолютний.
11.2.4. Системне властивість user.dir.
11.2.5. Основні методи File:

(1) робота з існуючим ЕФС;
(2) робота зі шляхами до ЕФС;
11.2.6. Отримання вмісту каталогу, фільтрація (File [name] Filer).
11.3. NIO (оглядово).
11.3.1. Класи Files, Paths, Path.
11.3.2. DirectoryStream.

Лекція № 12: Порівняння об'єктів.
12.1. Види рівності в Java.
12.2. Gорівняння об'єктів.
12.3. Методи equals, hashCode, їх контракти (+ зв'язок між ними), приклади з ядра.
12.4. Методи Comparator # compare, Comparable # compareTo, їх контракти, приклади з ядра.
12.5. Реалізація компараторов анонімними класами.
12.6. Зв'язок між Object # equals і Comparable # compareTo.

Лекція № 13: Колекції.
13.1. Ієрархія.
13.2. Інтерфейси Collection, List, Set, Queue.
13.3. Інтерфейси Iterator, Iterable.
13.4. Класи ArrayList, LinkedList, Vector, Stack.
13.5. Класи HashSet, TreeSet, критерій рівності елементів.
13.6. Способи проходження по контейнерах List, Set.

Лекція № 14: Асоціативні контейнери.
14.1. Ієрархія.
14.2. Інтерфейси Map, Map.Entry, їх методи.
14.3. Hash / Tree-контейнери HashMap, Hashtable, TreeMap.
14.4. Способи проходження по контейнеру.
14.5. Клас Properties, завантаження з файлів properties, xml, з classpath.
14.6. DTD на прикладі xml-файлів для Properties.
14.7. Класи Collections, Arrays, їх методи.

Лекція № 15: Багатопоточне програмування.
15.1. Базові поняття.
15.1.1. Термінологія.
(1) Потік як процес виконання команд.
(2) Потік як об'єкт.
15.1.2. Клас Thread і його методи (статичні / не статичною).
15.1.3. Головний потік.
15.1.4. Створення і запуск дочірнього потоку (Thread / Runnable).
15.1.5. Единственность способу запуску потоку.
15.1.6. Стану потоку (Thread.State).

15.1.7. Створення і запуск потоку в одному класі.
15.1.8. Запуск потоку в конструкторі класу потоку (pro / contra).
15.1.9. Створення і запуск потоку за допомогою анонімного класу.
15.1.10. Завершення виконання потоку.
15.1.11. Демони.
15.2. Взаємодія потоків.
15.2.1. Метод join.
15.2.2. Приклад одночасного доступу потоків до ресурсів.
15.2.3. Синхронізація (що може бути синхронізовано).
15.2.4. Монітор синхронізації.
15.2.5. Інструкція synchronized.
15.2.6. Методи wait / notify [all], їх спільне використання, приклад.
15.2.7. Блоковане стан пробудженого потоку.
15.2.8. Взаємне блокування.
15.2.9. Методи interrupt, interrupted, isInterrupted.
15.2.10. Приклад завершення одного потоку з іншого.
15.3. java.util.concurrent (оглядово).
15.3.1. Синхронізатори.
15.3.2. Ключове слово volatile, приклад.
15.3.3. Неблокіруючіх синхронізація (Lock).
15.3.4. Atomic-класи.
15.3.5. Приклад: швидкість роботи синхронізаторів.

3.4 Практичні заняття

Для кожного практичного заняття рекомендовано такий порядок підготовки:

1. Повторити теоретичний матеріал відповідно [4-6, див. п.2.4. "Рекомендована література"].

2. Відповісти на контрольні запитання по практичному заняттю.

3. Після проведення практичного заняття рекомендуємо виконати наступне: усі приклади, для виконання яких не вистачило часу, виконати самостійно.

4. На останньому практичному занятті проводиться комплексна контрольна робота (ККР).

Кожне практичне заняття необхідно захистити. Захист практичних занять виконується індивідуально кожним студентом.

3.5 Лабораторні роботи

Для кожної лабораторної роботи необхідно виконати такий порядок підготовки:

1. Повторити теоретичний матеріал відповідно [4-6, див. п.2.4. "Рекомендована література"].

2. Відповісти на контрольні запитання по лабораторній роботі.

3. Ознайомитися з порядком виконання роботи [4-6, див. п.2.4. "Рекомендована література"].

Кожну лабораторну роботу необхідно захистити. Захист лабораторних робіт виконується індивідуально кожним студентом.

4 Основні рекомендації з організації самостійної роботи

Для організації самостійної роботи необхідно використовувати учбові посібники з дисципліни "Основи програмування на Java". Обов’язкові розділи треба вивчати в порядку, в якому вони задані в програмі.

Навчання умовно поділяється на чотири основних етапи:

1) ознайомлювальний;

2) первинне засвоєння матеріалу;

3) накопичення інформації;

4) аналітичне осмислення і систематизація знань.

На першому етапі, який реалізовується на лекціях, здійснюється первинне знайомство з предметом вивчення, обсягом і змістом необхідних знань, складаються перші уявлення про систематизацію знань.

Другий етап це самостійне, індивідуальне опрацювання основних підручників і конспекту лекцій, спрямоване на отримання необхідних теоретичних знань. З початку рекомендується проглянути весь розділ, потім досконально розібрати всі пункти розділу, всі програми, які наведено, треба виконати на комп’ютері з отриманням та осмисленням результатів.

Третій етап реалізовується на лабораторних і практичних заняттях, в процесі підготовки, виконання, захисту завдань, розв’язання контрольних задач. На цьому етапі загальні теоретичні знання застосовують при розробці конкретних програм, поглиблюють в процесі виправки помилок, аналізу отриманих результатів. Тут корисне спілкування студентів між собою, проведення взаємних консультацій, дискусій, взаємної перевірки знань. Захист лабораторних і контрольних робіт здійснюється в присутності інших студентів. Завершальний етап аналітичне осмислення, аналіз і систематизація знань, набуття професійних навичок, знайомство зі спеціальною літературою, рекомендованою викладачем або вибраною відповідно до індивідуальних інтересів, самоконтроль знань в процесі підготовки відповідей на контрольні запитання і завдання, виконання контрольних робіт.

Така багатоступенева структура засвоєння сприяє отриманню глибоких і довготривалих знань, виробляє творче, аналітичне мислення. Для забезпечення регулярності виконання поточних завдань оцінка «відмінно» по кожному з завдань ставиться тільки в тому випадку, якщо завдання виконано в строк. Якщо завдання в строк не виконано без поважних причин (хвороба), то відмітка знижується на один бал.

Електронне навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ
до самостійної роботи з дисципліни
"ОСНОВИ ПРОГРАМУВАННЯ НА JAVA"

для студентів усіх форм навчання
напряму підготовки
6.050103 – "Програмна інженерія"

Упорядник: Колесников Дмитро Олегович.

Відповідальний за випуск
Редактор
Комп'ютерна верстка

Авторська редакція

План 2017 р. (), поз. 00.

Підп. до друку 00.00.2017 р. Формат 60×84$^{1}/_{16}$. Спосіб друку – ризографія.
Умов. друк. арк. 0,0. Облік. вид. арк. 0,0. Тираж 00 прим.
Зам. № 0-00. Ціна договірна.

ХНУРЕ, Україна, 61166, м. Харків, просп. Науки, 14

Надруковано в навчально-науковому видавничо-поліграфічному центрі
ХНУРЕ,
Україна, 61166, м. Харків, просп. Леніна, 14.

АЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

МЕТОДИЧНІ ВКАЗІВКИ
до практичних робіт з дисципліни
"ОСНОВИ ПРОГРАМУВАННЯ НА JAVA"

для студентів усіх форм навчання
напряму підготовки
6.050103 – "Програмна інженерія"

Електронне видання

Харків 2017

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

МЕТОДИЧНІ ВКАЗІВКИ
до практичних робіт з дисципліни
"ОСНОВИ ПРОГРАМУВАННЯ НА JAVA"

для студентів усіх форм навчання
напряму підготовки
6.050103 – "Програмна інженерія"

Електронне видання

ЗАТВЕРДЖЕНО
кафедрою ПІ.
Протокол № ___ від _____ р.

Харків 2017

Методичні вказівки до практичних робіт з дисципліни "Основи програмування на Java" для студентів усіх форм навчання першого (бакалаврського) рівня вищої освіти напряму підготовки 6.050103 - "Програмна інженерія" [Електронне видання] / Упоряд.: Д.О. Колесников. – Харків: ХНУРЕ, 2017. – 9 с.

Упорядник: Д.О. Колесников, доцент кафедри ПІ.

ЗМІСТ

ВСТУП

Метою практичних робіт з дисципліни "Основи програмування на Java" є закріплення теоретичного матеріалу та матеріалу, який вивчено самостійно.

Під час опрацювання теоретичного матеріалу студенти користуються загальною та практичною інформацією, яка наведена в даних методичних вказівках та яка рекомендована у спеціальній літературі.

Студенти закріплюють опрацьований матеріал відповідями на запитання для самоконтролю, які наведені в методичних вказівках.

Значну частину матеріалу студенти повинні вивчати самостійно з вказівок "Методичні вказівки до самостійної роботи з дисципліни "Розробка інтернет-застосувань".

Лекції призначені для вивчення головних теоретичних положень курсу. Обсяг лекцій не дозволяє вивчити всі розділи програми, тому частину теоретичних розділів студенти повинні вивчати самостійно. Для найкращого усвоєння теоретичного матеріалу рекомендується перед кожною лекцією прочитати матеріал попередньої лекції, а також рекомендований матеріал для самостійного вивчення.

1 Мета і задачі дисципліни

1.1 Мета дисципліни

Метою дисципліни "Основи програмування на Java" є є формування у студентів теоретичних знань та практичних навичок обєктно-орієнтованого програмування з використанням платформи Java Standard Edition.

1.2 Задачі дисципліни

За результатами вивчення дисциплін студенти мають вирішити такі задачі:

– вивчити мову програмування Java;

– ознайомитися з класами ядра Java;

– ознайомитися з принципами об'єктно-орієнтованої розробки застосувань мовою Java.

2 Класи і конструктори. Перетворення між ипами.

Мета роботи –ознайомитися і навчитися працювати з класами в Java.

2.1 Контрольні запитання

1. Які категорії типів даних існують в Java?
2. Перерахуйте примітивні типи даних.
3. Вкажіть автоматичні перетворення між примітивними типами.
4. Які перетворення між типами ви знаєте.
5. Що таке wrappers, boxing, unboxing.
6. Що таке перекриття методу, перевантаження.
7. Що таке поліморфізм, як його реалізувати.
8. Чи існує множинне успадкування в Java?
9. Які типи можуть бути успадковані?
10. Як написати метод зі змінним числом параметрів.
11. Що таке спадкування, ключові слова implements, extends.
12. Що таке інкапсуляція, для чого призначена, як реалізувати.
13. Ключове слово final, контексти використання.
14. Конструктори, їх призначення і відміну від методів.
15. Обмеження при перекритті методу.
16. Статичні методи і поля, відміну від нестатичних.
17. Блоки ініціалізації, які бувають.
18. Порядок виклику конструкторів, блоків ініціалізації при створенні об'єкта
19. Вкладені класи, анонімні класи, написати приклад.
20. У чому відмінність вкладених класів від внутрішніх.

2.2 Завдання

1. По книзі [6]:

a. прочитати глави 3 і 4 і пропрацювати тестові завдання в кінці глав;

b. зробити як мінімум по одному завданню з кожного варіанту, які дані в кінці кожного розділу.

2. По книзі [6]:

a. прочитати главу 6 і пропрацювати тестові завдання в кінці глави;

b. зробити як мінімум по одному завданню з кожного варіанту, які дані в кінці глави.

3 Строкові класи. Регулярні вирази. Інтернаціоналізація.

Ознайомитися і навчитися застосовувати основні інструментальні засоби Java призначені для роботи з текстовою інформацією.

3.1 Контрольні запитання

1. Строкові класи, відмінності, потокобезпечна.

2. Пул рядків, для чого потрібен. Інтернування рядків. Метод intern.
3. Регулярні вирази. Для чого потрібні. Написати приклад.
4. Pattern / Matcher. Чи є в класі String засоби для роботи з рег. виразами?
5. i18n / l10n. В чому різниця. Які засоби є для роботи з i18n.
6. Режими в регулярних виразах Java.
7. Попереджуючий перегляд вперед (позитивний, негативний).
8. Відмінність жодних регулярних вираженій від ледачих.
9. Які квантіфікатори в регулярних виразах ви знаєте.
10. Кодування за замовчуванням properties файлів.
11. Клас ResourceBundle.
12. Угруповання в регулярних виразах.

3.2 Завдання

1. За книгою [6] зробити по одній практичної задачі з кожного варіанту, які дані в кінці 7-го розділу.
2. Прочитати і засвоїти матеріал:
   a. http://skipy.ru/technics/strings.html
   b. http://skipy.ru/technics/localization.html

4 Generics. Анотації. Перечіслімий тип.

Мета роботи - ознайомлення з обобщением типів, методів і конструкторів; анотаціями та enum.

4.1 Контрольні запитання

1. Напишіть приклад generic типу, generic методу.
2. Що може бути узагальнене
3. Wildcards для generics (<?>, <? extends T>, <? super T>).
4. Параметри з обмеженнями (T extends A & B & C).
5. Чи можна наділити різною функціональністю елементи enum?
6. Який рівень доступу мають конструктори в enum?
7. Які статичні методи є у будь-якого enum?
8. Чи може enum реалізовувати інтерфейс, успадковувати клас або enum?
9. Коли необхідно параметризрвані узагальнений метод?
10. Що таке diamond оператор?

4.2 Завдання

1. Написати приклад узагальненого класу.
2. Написати приклад узагальненого методу.
3. Написати приклад власної анотації.
4. Написати приклад enum і клас, який йому відповідає.

5 Порівняння об'єктів.

Мета роботи – ознайомлення зі способами порівняння сутностей

5.1 Контрольні запитання

1. Оператор порівняння ==
2. Порівняння об'єктів на основі відношення еквівалентності.
3. Порівняння об'єктів на основі відношення порядку.
4. equals. Де визначено, призначення, контракт (5 пунктів).
5. Comparable / Comparator. Що містять, контракт методів.
6. hashCode. Де визначено. Контакт. Зв'язок з іншими методами.
7. Яким чином використовують hashCode. Як працює hash-таблиця.
8. Зв'язок між equals і compareTo / compare.
9. Що таке природне відношення порядку

5.2 Завдання

Написати приклад, який наочно ілюструє використання контейнерними класами ArrayList, HashSet, TreeSet (пакет java.util) різних алгоритмів визначення рівності об'єктів. Реалізувати власний клас, об'єкти якого ви будете додавати в контейнери. При додаванні об'єкта в той чи інший контейнер, має бути видно, які методи викликає контейнер на об'єкті вашого класу.

6 Асоціативні контейнери.

Мета роботи – ознайомлення з контейнерними класами.

6.1 Контрольні запитання

1. Lists. Які є, в чому відмінність.
2. Iterator / Iterable. У яких пакетах визначені, що містять.
3. Способи проходження по колекції (list / set). Написати приклади.
4. Sets. Які є, в чому відмінності.
5. Comparable / Comparator. Що містять, контракт методів.
6. Зв'язок між equals і compareTo / compare.
7. Maps. Інтерфейси, класи, в чому відмінність.
8. Способи проходження по контейнерах Map.

6.2 Завдання

Прочитати і засвоїти матеріал за наступними посиланнями:

http://skipy.ru/technics/objCompTh.html

http://skipy.ru/technics/objCompPr.html

Література

1. Шилдт Г. Java 8. Полное руководство : пер. с англ. / Г. Шилдт. – 9-е изд. – М.: Вильямс, 2015. – 1376 с.

2. The Java Language Specification. Java SE 8 Edition. [Електронний ресурс] http://docs.oracle.com/javase/specs/jls/se8/jls8.pdf

3. The Java Tutorials. [Електронний ресурс] http://docs.oracle.com/javase/tutorial

4. Хорстманн К., Корнелл Г. Java 2. Библиотека профессионала. Том первый. Основы. - М.: Вильямс, 2014. - 1008 с.

5. Хорстманн К., Корнелл Г. Java 2. Библиотека профессионала. Том второй. Тонкости программирования. - М.: Вильямс, 2014. - 864 с.

6. Java. Методы программирования : уч.-мет. пособие / И.Н. Блинов, В.С. Романчик. - Минск : издательство «Четыре четверти», 2013. — 896 с.

7. Конспект лекцій з дисципліни "Основи програмування на Java" для студентів усіх форм навчання напряму підготовки 6.050103 – "Програмна інженерія" [Електронний ресурс] / ХНУРЕ; Д.О. Колесников – Харків: ХНУРЕ, 2017. – 103 с. Електронна версія (pdf / 537 Kb).

8. Методичні вказівки до самостійної роботи з дисципліни "Основи програмування на Java" для студентів усіх форм навчання напряму підготовки 6.050103 – "Програмна інженерія" [Електронний ресурс] / ХНУРЕ; розроб. Д.О. Колесников. – Харків: ХНУРЕ, 2017. – 20 с. Електронна версія (pdf / 353 Kb).

9. Методичні вказівки до лабораторних робіт з дисципліни "Основи програмування на Java" для студентів усіх форм навчання напряму підготовки 6.050103 – "Програмна інженерія" [Електронний ресурс] / ХНУРЕ; розроб. Д.О. Колесников. – Харків: ХНУРЕ, 2017. – 20 с. Електронна версія (pdf / 445 Kb).

Електронне навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ
до практичних робіт з дисципліни
"ОСНОВИ ПРОГРАМУВАННЯ НА JAVA"

для студентів усіх форм навчання
напряму підготовки
6.050103 – "Програмна інженерія"

Упорядник: Колесников Дмитро Олегович.

Відповідальний за випуск
Редактор
Комп'ютерна верстка

Авторська редакція

План 2017 р. (), поз. 00.

Підп. до друку 00.00.2017 р. Формат 60×84 1/16. Спосіб друку – ризографія.
Умов. друк. арк. 0,0. Облік. вид. арк. 0,0. Тираж 00 прим.
Зам. № 0-00. Ціна договірна.

ХНУРЕ, Україна, 61166, м. Харків, просп. Науки, 14

Надруковано в навчально-науковому видавничо-поліграфічному центрі
ХНУРЕ,
Україна, 61166, м. Харків, просп. Леніна, 14.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт з дисципліни
"ОСНОВИ ПРОГРАМУВАННЯ НА JAVA"

для студентів усіх форм навчання
напряму підготовки
6.050103 – "Програмна інженерія"

Електронне видання

Харків 2017

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт з дисципліни
"ОСНОВИ ПРОГРАМУВАННЯ НА JAVA"

для студентів усіх форм навчання
напряму підготовки
6.050103 – "Програмна інженерія"

Електронне видання

ЗАТВЕРДЖЕНО
кафедрою ПІ.
Протокол № ___ від _____ р.

Харків 2017

Методичні вказівки до лабораторних робіт з дисципліни "Основи програмування на Java" для студентів усіх форм навчання першого (бакалаврського) рівня вищої освіти напряму підготовки 6.050103 - "Програмна інженерія" [Електронне видання] / Упоряд.: Д.О. Колесников – Харків: ХНУРЕ, 2017. – 20 с.

Упорядник: Д.О. Колесников, доцент кафедри ПІ.

ЗМІСТ

ВСТУП

Метою лабораторних робіт студентів з дисципліни "Основи програмування на Java" є закріплення на практиці теоретичного матеріалу за допомогою вирішення лабораторних задач.

Під час опрацювання теоретичного матеріалу студенти користуються загальною та практичною інформацією, яка наведена в даних методичних вказівках та яка рекомендована у спеціальній літературі.

Студенти закріплюють опрацьований матеріал відповідями на запитання для самоконтролю, які наведені в методичних вказівках. Самостійна робота над дисципліною має бути систематичною протягом всього періоду вивчення дисципліни.

Лекції призначені для вивчення головних теоретичних положень курсу. Обсяг лекцій не дозволяє вивчити всі розділи програми, тому частину теоретичних розділів студенти повинні вивчати самостійно. Для найкращого усвоєння теоретичного матеріалу рекомендується перед кожною лекцією прочитати матеріал попередньої лекції, а також рекомендований матеріал для самостійного вивчення.

Кожна лабораторна робота захищається кожним студентом індивідуально.

1 Мета і задачі дисципліни

1.1 Мета дисципліни

Метою дисципліни "Основи програмування на Java" є є формування у студентів теоретичних знань та практичних навичок об'єктно-орієнтованого програмування з використанням платформи Java Standard Edition.

1.2 Задачі дисципліни

За результатами вивчення дисциплін студенти мають вирішити такі задачі:

- вивчити мову програмування Java;
- ознайомитися з класами ядра Java;
- ознайомитися з принципами об'єктно-орієнтованої розробки застосувань мовою Java.

2 Робота з операторами та масивами

2.1 Мета роботи

Ознайомитися і навчитися працювати з основними операторами і операціями мови Java, а також масивами.

2.2 Необхідне ПЗ

Java 8

2.3 Установка оточення

Для виконання лабораторної роботи необхідно встановити JDK 8. Доцільно використовувати IDE (Eclipse, IntelliJ IDEA).

Кореневої пакет для всіх Java-класів:

```
ua.nure.yourlastname.task1
```

де yourlastname - ваше прізвище в англійській транслітерації в нижньому регістрі.

2.4 Завдання

У всіх завданнях вхідна інформація надходить у вигляді параметрів командного рядка. Вихідна інформація, яку генерує завдання, надходить в стандартний потік виводу (консоль).

2.4.1 Завдання 1

```
Назва класу: ua.nure.yourlastname.task1.Part1
Вхід: X Y
Вихід: найбільший спільний дільник X і Y
```

Написати програму, яка знаходить суму цифр заданого цілого числа.

2.4.2 Завдання 2

```
Назва класу: ua.nure.yourlastname.task1.Part2
Вхід: X
Вихід: сума цифр числа X
```

Написати програму, яка знаходить суму цифр заданого цілого числа.

2.4.3 Завдання 3

```
Назва класса: ua.nure.yourlastname.task1.Part3
Вхід: X
Вихід: true, если X простое число; false в противном случае
```

Написати програму перевірки того, що заданий число просте (просте число - натуральне число більше одиниці, яке має тільки два цілих подільника - 1 і саме це число).

2.4.4 Завдання 4

```
Название класса: ua.nure.yourlastname.task1.Part4
Вхід: N
Вихід: сумма ряда 1! - 2! + 3! - 4! + 5! - ... +/- N!
```

Підрахувати суму ряду для заданого числа N> 0:

```
1! - 2! + 3! - 4! + 5! - ... +/- N!
```

2.4.5 Задание 5

```
Название класса: ua.nure.yourlastname.task1.Part5
Вхід: N
Вихід: количество N-значных "счастливых" чисел
```

Підрахувати, скільки N-значних чисел мають рівну суму першої половини і другої половини цифр ("щасливі" числа). Число N є парним позитивним числом.

Приклад. Якщо N = 6, то результат (вихід) представляє з себе кількість шестизначних чисел ABCDEF (провідний нуль не допускається), для яких:

```
(A+B+C) = (D+E+F)
```

2.4.6 Завдання 6

```
Назва класу: ua.nure.yourlastname.task1.Part6
Вхід: N
Вихід: первые N элементов ряда Фибоначчи
```

Розмістити у пам'яті масив з N елементів і заповнити його поруч Фібоначчі: 1, 1, 2, 3, 5, 8, 13, 21, ... У цьому ряду кожне наступне число є сумою двох попередніх.

2.4.7 Завдання 7

```
Назва класу: ua.nure.yourlastname.task1.Part7
Вхід: N
Вихід: первые N простых чисел в ряде натуральных чисел,
разделенные пробелами
```

Створити цілий масив з N елементів і заповнити його простими числами: 2, 3, 5, 7, 11, 13, 17. Приклад. Якщо N = 5, то результатом є масив з елементами 2 3 5 7 11

2.4.8 Завдання 8

```
Назва класу: ua.nure.yourlastname.task1.Part8
Вхід: N M
Вихід: таблица с буквами 'Ч' и 'Б' в шахматном порядке
```

Створити двовимірний масив (N * M) символів і заповнити його буквами 'Ч' і 'Б' в шаховому порядку.

Приклад. Якщо N = 3, M = 5, то результат повинен представляти із себе такбліцу:

```
ЧБЧБЧ
БЧБЧБ
ЧБЧБЧ
```

2.4.9 Завдання 9

```
Назва класу: ua.nure.yourlastname.task1.Part9
Вхід: отсутствует
Вихід: 1 2 3 ... 32
```

Створити цілий пятімерний масив з двома значеннями в кожному вимірі. Заповнити масив числами з початку натурального ряду: 1, 2, 3, ... При заповненні використовувати один цикл.

2.4.10 Завдання 10

```
Назва класу: ua.nure.yourlastname.task1.Part10
Вхід: N
Вихід: биномиальные коэффициенты разделенные пробелами
```

Створити "трикутний" масив з N рядків і заповнити його біноміальними коефіцієнтами.

Приклад. Якщо N = 5, то результат повинен представляти із себе таблицю:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

2.5 Зауваження

1. Проект в IDE (Eclipse, IDEA) повинен мати назву Task1.
2. Рішення кожної з 10 завдань оформити у вигляді окремого класу з назвами - Part1, Part2, ..., Part10. Значення вхідних даних отримувати з командного рядка. У разі, якщо вхідні дані задано не були - виводити інформаційне повідомлення про це.
3. Додатково в кореневому пакеті створити клас Demo, який демонструє роботу всіх 10 підпрограм з деякими заданими значеннями (задати самостійно).
4. Додатково написати командний файл (bat-файл в Windows або shell-скрипт в Linux), який компілює і запускає на виконання клас Demo. Скрипт зберегти в каталозі Task1 / src. Вручну запустити скрипт, перевірити його работоспособность.
5. Дотримуватися code convention.

2.6 Хід виконання роботи

1. Написати класи, які реалізують рішення задач 1-10. Написати клас Demo.
2. Створити командний файл для вашої OS.
3. Прив'язати проект до потрібного вузла в репозиторії і зробити Комміт проекту в репозиторій.
4. Комміт не пройде, якщо серед файлів, які розміщені в репозиторій, буде хоча б один class-файл, таким чином слід class-файли з коміта виключити.
5. В обов'язковому порядку в репозиторій повинні бути поміщені метафайли IDE (для Eclipse файл .project; для IDEA файл Task1.iml).

6. Домогтися збірки проекту в Jenkins (після кожного коммітов в репозиторій Jenkins збирати заново проект, якщо проект не був зібраний, то причину можна подивитися в логах збірки).

7. Оптимізувати метрики проекту в Sonar (кількість Blocker / Critical / Major isuues має бути 0, метрика RCI якомога ближче до 100%).

2.7 Контрольні питання

1. Які категорії типів даних існують в Java?
2. Перерахуйте примітивні типи даних.
3. Тип даних char, що зберігає, область визначення.
4. Напишіть метод main (два варіанти).
5. Вкажіть автоматичні перетворення між примітивними типами
6. За яких перетвореннях між примітивними типами можлива втрата інформації?
7. Які перетворення між типами ви знаєте.
8. Що таке wrappers, autoboxing, autounboxing.
9. Напишіть анонімний масив, масив константу, в чому відмінність.
10. Напишіть приклад двовимірного масиву.

3 Робота з класами

3.1 Мета роботи

Ознайомитися і навчитися працювати з класами в Java.

3.2 Необхідне ПЗ

Java 8

3.3 Установка оточення

Для виконання лабораторної роботи необхідно встановити JDK 8. Доцільно використовувати IDE (Eclipse, IntelliJ IDEA).

Кореневої пакет для всіх Java-класів:

```
ua.nure.yourlastname.task2
```

де yourlastname - ваше прізвище в англійській транслітерації в нижньому регістрі.

3.4 Завдання

У кожному з трьох класів (Circle, Matrix, ListImpl) повинен бути метод main, який демонструє роботу відповідного класу. Всі дані задавати всередині методів main.

3.4.1 Завдання 1

```
Назва класу: ua.nure.yourlastname.task2.Circle
```

Реалізувати клас Circle, який поставлений у відповідність геометричної фігури коло.

Поля класу. Координати центру кола:

```
double x
double y
```

Радіус кола:

```
double r
```

Методи класу:

```
// передвинуть окружность на dx и dy:
public void move(double dx, double dy) { ... }

// проверить попадание заданной точки
// внутрь данной окружности:
public boolean isInside(double x, double y) { ... }

// проверить попадание другой окружности внутрь данной
public boolean isInside(Circle c) { ... }

// вывести на экран параметры окружности
public void print() { ... }
```

Конструктори:

```
public Cirlce(double x, double y, double r) { ... }
```

Створити метод main, який демонструє роботу всіх методів класу.

3.4.2 Завдання 2

```
Назва класу: ua.nure.yourlastname.task2.Matrix
```

Реалізувати клас, який поставлений у відповідність такому математичного об'єкту, як матриця.

Поля класу. Двовимірний масив дійсних чисел, який зберігає елементи матриці:

```
double[rows][cols] ar
```

Кількість рядків і стовпців в матриці.

```
int rows
int cols
```

Методи класу:

```
// сложение с другой матрицей:
public void add(Matrix m) { ... }

// умножение на число:
public void mul(double x) { ... }

// умножение на другую матрицу:
public void mul(Matrix m) { ... }

// транспонирование:
public void transpose() { ... }

// печать матрицы на экран:
public void print() { ... }
```

Конструктори:

```
public Matrix(double[][] ar) { ... }
```

Создать метод main, который демонстрирует работу всех методов класса.

3.4.3 Завдання 3

```
Назва класу: ua.nure.yourlastname.task2.ListImpl
```

1. Визначити інтерфейс List такого змісту:

```
public interface List {
    // appends the specified element
    // to the end of this list
    void add(Object el);

    // appends all of the elements
    // in the specified collection to the end of this list
    void addAll(List list);

    // removes all of the elements from this list
    void clear();

    // returns true if
    // this list contains the specified element
    boolean contains(Object el);

    // returns the element
    // at the specified position in this list.
    Object get(int index);

    // returns the index of the first occurrence
    // of the specified element in this list
    int indexOf(Object el);

    // removes the element at the specified position
    // in this list, returns the element previously
    // at the specified position
    // throws IndexOutOfBoundsException
```

```
    // if the index is out of range
    Object remove(int index);

    // removes the first occurrence
    // of the specified element from this list,
    // returns true if this list contained
    // the specified element
    boolean remove(Object el);

    // returns the number of elements in this list
    int size();
}
```

2. Створити клас ListImpl, який реалізує інтерфейс List:

```
public class ListImpl implements List {
    public ListImpl() { ... }
    ...
}
```

3. Визначити інтерфейс Iterator, успадкувавши його від java.util.Iterator <Object>:

```
public interface Iterator extends java.util.Iterator<Object> {

    // returns true if the iteration has more elements
    boolean hasNext();

    // returns the next element in the iteration
    E next();

    // removes the last element returned by this iterator
    // this method can be called only once per call to next()
    // method throws IllegalStateException if:
    // (1) the next method has not yet been called, or
    // (2) the remove method has already been called
    // after the last call to the next method
    void remove();
}
```

4. Додати до інтерфейсу List успадкування інтерфейсу java.lang.Iterable <Object>:

```
public interface List extends java.lang.Iterable<Object> {
     ...
    // returns an iterator
    Iterator iterator();
}
```

5. Реалізувати метод iterator в класі ListImpl.

6. Перекрити в класі ListImpl метод toString, в такий спосіб. Якщо list містить елементи E, E2, ..., En, то list.toString повертає рядок

```
[E.toString(), E2.toString(), ..., En.toString()]
```

7. Створити метод main, який демонструє роботу всіх методів класу ListImpl.

3.5 Зауваження

1. Проект в IDE (Eclipse, IDEA) повинен мати назву Task2.
2. Всі типи повинні знаходиться в кореневому пакеті.
3. Додатково повинен бути клас Demo, який демонструє роботу всіх класів.
4. Дотримуватися code convention.

3.6 Хід виконання роботи

1. Створити інтерфейси і реалізувати всі класи, які потрібні.
2. Написати клас Demo.
3. В обов'язковому порядку в репозиторій повинні бути поміщені метафайли IDE (для Eclipse файл .project; для IDEA файл Task2.iml).
4. Домогтися збірки проекту в Jenkins (після кожного коммітов в репозиторій Jenkins збирати заново проект, якщо проект не був зібраний, то причину можна подивитися в логах збірки).
5. Оптимізувати метрики проекту в Sonar (кількість Blocker / Critical / Major isuues має бути 0, метрика RCI якомога ближче до 100%).

3.7 Контрольні питання

1. Що таке перекриття методу, перевантаження.
2. Що таке поліморфізм, як його реалізувати.
3. Чи існує множинне успадкування в Java?
4. Які типи можуть бути успадковані?
5. Що таке спадкування, ключові слова implements, extends.
6. Що таке інкапсуляція, для чого призначена, як реалізувати.
7. Ключове слово final, контекст використання.
8. Конструктори, їх призначення і відміну від методів.
9. Обмеження при перекритті методу.
10. Блоки ініціалізації, які бувають.
11. Вкажіть порядок виклику конструкторів, блоків ініціалізації при створенні об'єкта.
12. Вкладені класи, анонімні класи, написати приклад.
13. У чому відмінність вкладених класів від внутрішніх.

4 Робота з рядками

4.1 Мета роботи

Ознайомитися і навчитися застосовувати основні інструментальні засоби Java призначені для роботи з текстовою інформацією.

4.2 Необхідне ПЗ

Java 8

4.3 Установка оточення

Для виконання лабораторної роботи необхідно встановити JDK 8. Доцільно використовувати IDE (Eclipse, IntelliJ IDEA).

Кореневої пакет для всіх Java-класів:

```
ua.nure.yourlastname.task3
```

де yourlastname - ваше прізвище в англійській транслітерації в нижньому регістрі.

4.4 Завдання

Якщо додаток зчитує інформацію з файлу, то необхідно вказати кодування, в якій вона (інформація) записана. Якщо явно не вказано мову, на якому записана текстова інформація, то текст повинен містити кирилицю (російська, українська і т.п.), додатково текст може містити латиницю.

Для вирішення завдань використовувати регулярні вирази. Не допускається використання контейнерних класів.

4.4.1 Завдання 1

```
Назва класу: ua.nure.yourlastname.task3.Part1
Вхідну інформацію завантажувати з файлу part1.txt
```

Створити клас, який виводить вміст текстового файлу в консоль, інвертуємо регістр символів у всіх словах, які складаються з трьох і більше символів. Файл брати розміром не більше 1 Кб (досить кілька рядків).

4.4.2 Завдання 2

```
Назва класу: ua.nure.yourlastname.task3.Part2
Вхідну інформацію завантажувати з файлу part2.txt
```

Створити клас, який виводить вміст текстового файлу в консоль, видаляючи всі слова, які містять однакові букви в своєму складі. Під словом розуміти безперервну послідовність непробельний символів.

4.4.3 Завдання 3

```
Назва класу: ua.nure.yourlastname.task3.Part3
Вхідну інформацію завантажувати з файлу part3.txt
```

Створити клас, який виводить вміст текстового файлу в консоль, видаляючи всі слова, які мають дублікати в тексті. Під словом розуміти безперервну послідовність непробельний символів.

4.4.4 Завдання 4

```
Назва класу: ua.nure.yourlastname.task3.Part4
Вхідну інформацію завантажувати з файлу part4.txt
```

Файл містить слова, цілі і речові числа розділені пробілами. Написати клас, який реалізує наступну функціональність: в циклі користувач вводить тип даних (string, int, double), у відповідь додаток друкує в консоль все значення відповідного типу, які існують в файлі (виводяться значення повинні бути в тому порядку, в якому вони зустрічаються в вихідному файлі, значення розділяти пробілом).

Ознакою закінчення введення служить слово stop.

Приклад. Якщо вміст файлу part4.txt таке:

```
яd ffa bcd ы 43.43 432 15. и л фвыа 89 .98
```

То результат (включет консольне введення і виведення) повинен бути таким:

```
string
яd ffa bcd ы и л фвыа
int
432 89
double
43.43 15. .98
stop
```

4.4.5 Завдання 5

```
Назва класу: ua.nure.yourlastname.task3.Part5
Вхідну інформацію завантажувати з файлу part5.txt
```

Визначити клас зі статичними методами, які перетворюють вхідну інформацію в вихідну. В якості вхідної інформації (input data) використовувати текст наступної структури (значення Login/Name/Email в загальному випадку можуть бути будь-якими):

```
Login;Name;Email
ivanov;Ivan Ivanov;ivanov@mail.ru
петров;Петр Петров;petrov@google.com
lennon;John Lennon;lennon@google.com
```

Методи, які потрібно написати, мають такий вигляд (N - цифра: 1, 2, 3, 4):

```
public static String convertN(String input) {
    ...
}
```

4.4.5.1 Метод convert1

Повинен перетворювати input data в рядок наступного виду:

```
ivanov ==> ivanov@mail.ru
петров ==> petrov@google.com
lennon ==> lennon@google.com
```

4.4.5.2 Метод convert2

Повинен перетворювати вхідні дані в рядок наступного виду:

```
Ivan Ivanov (email: ivanov@mail.ru)
Петр Петров (email: petrov@google.com)
John Lennon (email: lennon@google.com)
```

4.4.5.3 Метод convert3

Повинен перетворювати вхідні дані в рядок наступного виду:

```
mail.ru ==> ivanov
google.com ==> петров, lennon
```

(Поштовий домен ==> список логінів через кому тих користувачів, чиї поштові скриньки зарєестрованих в даному домені)

4.5 Хід виконання роботи

1. Реалізувати необхідну функціональність.

2. У кореневому пакеті створити клас Demo, який демонструє роботу всього написаного функціоналу.

3. В обов'язковому порядку в репозиторій повинні бути поміщені метафайли IDE (для Eclipse файл .project; для IDEA файл Task3.iml).

4. Домогтися збірки проекту в Jenkins (після кожного коммітов в репозиторій Jenkins збирати заново проект, якщо проект не був зібраний, то причину можна подивитися в логах збірки).

5. Оптимізувати метрики проекту в Sonar (кількість Blocker / Critical / Major isuues має бути 0, метрика RCI якомога ближче до 100%).

4.6 Контрольні питання

1. Регулярні вирази. Для чого потрібні. Написати приклад.

2. Pattern / Matcher. Чи є в класі String кошти для роботи з регулярними. виразами, які?

3. Прапори в регулярних виразах Java.

4. Попереджуючий перегляд вперед (позитивний, негативний).

5. Відмінність жодних регулярнихвираженій від ледачих.

6. Які квантіфікатори в регулярних виразах ви знаєте.

7. Кодування за замовчуванням properties файлів.

8. Клас ResourceBundle для чого призначений.

9. Відмінності між класами String, StringBuilder, StringBuffer.

10. Угруповання в регулярних виразах.

5 Робота з потоками введення виведення і контейнерними класами

5.1 Мета роботи

Ознайомитися і навчитися працювати з інструментальними засобами Java для роботи з потоками введення виведення і контейнерними класами.

5.2 Необхідне ПЗ

Java 8

5.3 Установка оточення

Для виконання лабораторної роботи необхідно встановити JDK 8. Доцільно використовувати IDE (Eclipse, IntelliJ IDEA).

Кореневої пакет для всіх Java-класів:

```
ua.nure.yourlastname.task4
```

де yourlastname - ваше прізвище в англійській транслітерації в нижньому регістрі.

5.4 Завдання

Якщо додаток зчитує інформацію з файлу, то необхідно вказати кодування, в якій вона (інформація) записана. Якщо явно не вказано мову, на якому записана текстова інформація, то текст повинен містити кирилицю (російська, українська і т.п.), додатково текст може містити латиницю.

5.4.1 Завдання 1

```
Назва класу: ua.nure.yourlastname.task4.Part1
Вхідну інформацію завантажувати з файлу part1.txt
```

Створити клас, який реалізує інтерфейс java.lang.Iterable.

Клас повинен розбирати текстовий файл і повертати слова з файлу (в тому порядку, в якому вони зустрічаються в вихідному файлі). Під словами розуміти безперервну послідовність, що складається з літер російського, українського та англійського алфавітів в верхньому і нижньому регістрах.

Вихідний файл брати невеликим за розміром, досить кілька рядків (як мінімум два рядки). Не допускається використовувати існуючі реалізації ітераторів з контейнерних класів. Слід використовувати регулярні вирази. У методі Iterator#remove прописати викид виключення UnsupportedOperationException.

Рекомендація. Напишіть регулярний вираз, яке "вирізає" потрібні послідовності з тексту, далі використовуйте об'єкт Matcher при реалізації методів інтерфейсу Iterator.

5.4.2 Завдання 2

```
Назва класу: ua.nure.yourlastname.task4.Part2
Вхідну інформацію завантажувати з файлу part2.txt
```

Створити клас, який зчитує слова з файлу і виводить їх в консоль в порядку убування частоти їх появи в тексті (при збігу частот порядок - лексікографічекій).

Вирішити завдання із застосуванням ООП підходу: контейнер WordContainer агрегує об'єкти Word (клас Word містить строкове поле content і ціле поле frequency).

При використанні контейерних класів ядра грамотно реалізувати методи Word#equals / Word#hashCode / Word#compareTo якщо вони будуть потрібні.

5.4.3 Завдання 3

```
Назва класу: ua.nure.yourlastname.task4.Part3
Ресурси:resources_en.properties, resources_ru.properties
```

Створити пакети ресурсів (properties файли) для двох локалізацій: ru і en (розмістити їх в каталозі src). Варіанти комплектації включають як мінімум два записи, наприклад:

```
table = table
apple = apple
```

в файлі resources_en.properties и

```
table = стол
apple = яблоко
```

в файлі resources_ru.properties.

Написати клас, який в циклі читає з консолі ключ (key) і ім'я локалізації через пробіл, у відповідь друкує відповідне значення в консоль.

Читання з консолі і запис в консоль є обов'язковими. Ознакою закінчення введення є слово stop.

5.4.4 Завдання 4

```
Назва класу: ua.nure.yourlastname.task4.Part4
Вхідна інформація: файл part4.txt
Вихідна інформація: файл part4_sorted.txt
```

Вміст методу Part4.main має бути таким:

```
public static void main(String[] args) {
    generate();
    process();
    print();
}
```

5.4.4.1 Метод generate

Написати статичний метод, який створює і заповнює текстовий файл part4.txt випадковими цілими числами від 0 до 50 (всього 10 чисел). Файл повинен бути текстовим (читабельним) і містити числа, розділені пропуском, кодування файлу - UTF-8. Метод повинен мати ім'я generate і не повинен приймати параметри:

```
public static void generate() {...}
```

5.4.4.2 Метод process

Написати статичний метод, який читає файл part4.txt і виводить його вміст в файл part4_sorted.txt, відсортувавши числа по зростанню. Метод повинен мати ім'я process і не повинен приймати параметри:

```
public static void process() {...}
```

Для сортування написати власний метод, який здійснює сортування деяким алгоритмом (наприклад "бульбашкою"). Вихідний файл повинен бути текстовим (читабельним) в кодуванні UTF-8.

5.4.4.3 Метод print

Написати статичний метод, який виводить вміст обох файлів (числа розділені пробілом) в консоль (всього буде два рядки). Формат виведення показаний на наступному прикладі (зверніть увагу на те, що зліва і праворуч від ==> знаходиться рівно один пробіл):

```
input ==> 30 23 16 16 9 23 3 18 21 29
output ==> 3 9 16 16 18 21 23 23 29 30
```

Метод повинен мати ім'я process і не повинен приймати параметри:

```
public static void print() {...}
```

5.4.5 Завдання 5

```
Назва класу: ua.nure.yourlastname.task4.Part5
Вхідна інформація: файл part5.txt
```

У вхідному файлі знайти три слова, які зустрічаються найбільш часто (при збігу частот - ті, які зустрічаються раніше), і роздрукувати їх відсортованими за алфавітом у зворотному порядку в форматі: слово ==> частота.

Приклад виведення:

```
whale ==> 3
rabbit ==> 4
bison ==> 3
```

5.5 Зауваження

У кореневому пакеті повинен знаходиться клас Demo, який демонструє роботу всього функціоналу.

Для тих подзадач, які вимагають введення з консолі, необхідно цей введення моделювати. перепризначити стандартний потік введення таким чином, щоб введення здійснювався з деякою заданою рядки. Demo.main повинен відпрацьовувати без участі користувача, ніякого очікування введення з консолі при виконанні даного методу бути не повинно.

5.6 Хід виконання роботи

1. Реалізувати необхідну функціональність.
2. У кореневому пакеті створити клас Demo, який демонструє роботу всього написаного функціоналу.
3. В обов'язковому порядку в репозиторій повинні бути поміщені метафайли IDE (для Eclipse файл .project; для IDEA файл Task4.iml).
4. Домогтися збірки проекту в Jenkins (після кожного коммітов в репозиторій Jenkins збирати заново проект, якщо проект не був зібраний, то причину можна подивитися в логах збірки).
5. Оптимізувати метрики проекту в Sonar (кількість Blocker / Critical / Major isuues має бути 0, метрика RCI якомога ближче до 100%).

5.7 Контрольні питання

1. Потоки введення / виведення, класифікація, привести приклади класів.
2. InputStreamReader / OutputStreamWriter.
3. Для чого ввели дві категорії потоків - символьні і байтові.
4. Кодування, приклади кодувань, кодування за замовчуванням.
5. Unicode. UTF. Види, відмінності. BE / LE варіанти. BOM.
6. Колекції. Ієрархія інтерфейсів, класів. Collections.
7. Lists. Які є, в чому відмінність.
8. Iterator / Iterable. У яких пакетах визначені, що містять.
9. Способи проходження по колекції (list / set). Написати приклади.
10. Sets. Які є, в чому відмінності.

6 Робота з потоками виконання

6.1 Мета роботи

Ознайомитися і навчитися працювати з інструментальними засобами Java для роботи з потоками виконання.

6.2 Необхідне ПЗ

Java 8

6.3 Установка оточення

Для виконання лабораторної роботи необхідно встановити JDK 8. Доцільно використовувати IDE (Eclipse, IntelliJ IDEA).

Кореневої пакет для всіх Java-класів:

```
ua.nure.yourlastname.task5
```

де yourlastname - ваше прізвище в англійській транслітерації в нижньому регістрі.

6.4 Завдання

6.4.1 Завдання 1

```
Назви класів:
ua.nure.yourlastname.task5.Part1
ua.nure.yourlastname.task5.Spam
```

Створити клас Spam, який отримує масив інтервалів часу в мілісекундах і узгоджений з ним масив повідомлень, і виводить одночасно відповідні повідомлення на екран через задані інтервали часу.

При натисканні на клавішу Enter додаток повинен завершувати свою роботу (дану функціональність помістити в метод Spam.main).

При демонстрації роботи змоделювати введення Enter через 3 сек (дану функціональність помістити в метод Part2.main).

6.4.2 Завдання 2

```
Назва класу: ua.nure.yourlastname.task5.Part2
Вихідну інформацію записувати в файл part2.txt
```

Створити десять потоків, які одночасно пишуть в один і той же файл цифри:

```
перший потік пише цифру 0 рівно 20 разів на 1й рядку файлу;
другий потік пише цифру 1 рівно 20 разів на 2й рядку файлу;
...
дев'ятий потік пише цифру 9 рівно 20 разів на 10-му рядку файлу.
```

Для запису використовувати клас RandomAccessFile (допускається використання не більше одного об'єкта цього класу). Перед початком роботи файл в який відбуватиметься запис повинен бути вилучений, якщо він існує. Головний потік, після запуску дочерніх потоків на виконання, повинен дочекатися їх завершення, після чого вивести в консоль вміст файлу.

Приклад виведення:

```
00000000000000000000
11111111111111111111
22222222222222222222
33333333333333333333
44444444444444444444
```

```
5555555555555555555
6666666666666666666
7777777777777777777
8888888888888888888
9999999999999999999
```

6.5 Зауваження

У кореневому пакеті повинен знаходиться клас Demo, який демонструє роботу всього функціоналу. Demo.main повинен відпрацьовувати без участі користувача, ніякого очікування введення з консолі при виконанні даного методу бути не повинно.

6.6 Хід виконання роботи

1. Реалізувати необхідну функціональність.

2. У кореневому пакеті створити клас Demo, який демонструє роботу всього написаного функціоналу.

3. В обов'язковому порядку в репозиторій повинні бути поміщені метафайли IDE (для Eclipse файл .project; для IDEA файл Task5.iml).

4. Домогтися збірки проекту в Jenkins (після кожного коммітов в репозиторій Jenkins збирати заново проект, якщо проект не був зібраний, то причину можна подивитися в логах збірки).

5. Оптимізувати метрики проекту в Sonar (кількість Blocker / Critical / Major isuues має бути 0, метрика RCI якомога ближче до 100%).

6.7 Контрольні питання

1. Потоки виконання. Створення. Для чого потрібен Runnable.

2. Життєвий цикл потоку. Коли закінчує своє виконання.

3. Потоки демони, властивості, для чого потрібні, як зробити потік демоном.

4. Синхронізація. Для чого потрібна.

5. Що може бути синхронізовано.

6. Монітор синхронізації і його роль в розмежуванні доступу до синхрон. коду.

7. Методи sleep / wait. В чому різниця.

8. Фантомні пробудження.

9. Метод join.

10. Методи interrupt, interrupted, isInterrupted.

11. Як отримати поточний потік.

12. Які методи знімають блокування з монітора.

Література

1. Шилдт Г. Java 8. Полное руководство : пер. с англ. / Г. Шилдт. – 9-е изд. – М.: Вильямс, 2015. – 1376 с.

2. The Java Language Specification. Java SE 8 Edition. [Електронний ресурс] http://docs.oracle.com/javase/specs/jls/se8/jls8.pdf

3. The Java Tutorials. [Електронний ресурс] http://docs.oracle.com/javase/tutorial

4. Хорстманн К., Корнелл Г. Java 2. Библиотека профессионала. Том первый. Основы. - М.: Вильямс, 2014. - 1008 с.

5. Хорстманн К., Корнелл Г. Java 2. Библиотека профессионала. Том второй. Тонкости программирования. - М.: Вильямс, 2014. - 864 с.

6. Конспект лекцій з дисципліни "Основи програмування на Java" для студентів усіх форм навчання напряму підготовки 6.050103 – "Програмна інженерія" [Електронний ресурс] / ХНУРЕ; Д.О. Колесников – Харків: ХНУРЕ, 2017. – 103 с. Електронна версія (pdf / 537 Kb).

7. Методичні вказівки до практичних робіт з дисципліни "Основи програмування на Java" для студентів усіх форм навчання напряму підготовки 6.050103 – "Програмна інженерія" [Електронний ресурс] / ХНУРЕ; розроб. Д.О. Колесников. – Харків: ХНУРЕ, 2017. – 8 с. Електронна версія (pdf / 333 Kb).

8. Методичні вказівки до самостійної роботи з дисципліни "Основи програмування на Java" для студентів усіх форм навчання напряму підготовки 6.050103 – "Програмна інженерія" [Електронний ресурс] / ХНУРЕ; розроб. Д.О. Колесников. – Харків: ХНУРЕ, 2017. – 20 с. Електронна версія (pdf / 353 Kb).

Електронне навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт з дисципліни
"ОСНОВИ ПРОГРАМУВАННЯ НА JAVA"

для студентів усіх форм навчання
напряму підготовки
6.050103 – "Програмна інженерія"

Упорядник: Колесников Дмитро Олегович.

Відповідальний за випуск
Редактор
Комп'ютерна верстка

Авторська редакція

План 2017 р. (), поз. 00.

Підп. до друку 00.00.2017 р. Формат 60×84 1/16. Спосіб друку – ризографія.
Умов. друк. арк. 0,0. Облік. вид. арк. 0,0. Тираж 00 прим.
Зам. № 0-00. Ціна договірна.

ХНУРЕ, Україна, 61166, м. Харків, просп. Науки, 14

Надруковано в навчально-науковому видавничо-поліграфічному центрі
ХНУРЕ,
Україна, 61166, м. Харків, просп. Леніна, 14.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

ПЕРЕЛІК ЗАПИТАНЬ
ДО ПІДСУМКОВОГО МОДУЛЬНОГО КОНТРОЛЮ

з дисципліни
"ОСНОВИ ПРОГРАМУВАННЯ НА JAVA"

для студентів усіх форм навчання
напряму підготовки
6.050103 – "Програмна інженерія"

Електронне видання

Харків 2017

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

ПЕРЕЛІК ЗАПИТАНЬ
ДО ПІДСУМКОВОГО МОДУЛЬНОГО КОНТРОЛЮ

з дисципліни
"ОСНОВИ ПРОГРАМУВАННЯ НА JAVA"

для студентів усіх форм навчання
напряму підготовки
6.050103 – "Програмна інженерія"

Електронне видання

ЗАТВЕРДЖЕНО
кафедрою ПІ.
Протокол № ___ від _____ р.

Харків 2017

Перелік запитань до підсумкового модульного контролю з дисципліни "Основи програмування на Java" для студентів усіх форм навчання першого (бакалаврського) рівня вищої освіти напряму підготовки 6.050103 - "Програмна інженерія" [Електронне видання] / Упоряд.: Д.О. Колесников. – Харків: ХНУРЕ, 2017. – 4 с.

Упорядник: Д.О. Колесников, доцент кафедри ПІ.

ПЕРЕЛІК ЗАПИТАНЬ ДО
ПІДСУМКОВОГО МОДУЛЬНОГО КОНТРОЛЮ
з дисципліни "Основи програмування на Java "

1. Які категорії типів даних існують в Java?
2. Перерахуйте примітивні типи даних.
3. Тип даних char, область визначення.
4. Напишіть метод main.
5. Вкажіть автоматичні перетворення між примітивними типами.
6. Які перетворення між типами ви знаєте.
7. Що таке wrappers, boxing, unboxing.
8. Напишіть анонімний масив, масив константу, в чому відмінність.
9. Напишіть приклад двовимірного масиву.
10. Що таке перекриття методу, перевантаження.
11. Що таке поліморфізм, як його реалізувати.
12. Чи існує множинне успадкування в Java?
13. Які типи можуть бути успадковані?
14. Як написати метод зі змінним числом параметрів.
15. Напишіть приклад generic типу, generic методу.
16. Які суті можуть бути generic.
17. Wildcards для generics (<?>, <? Extends T>, <? Super T>).
18. Параметри з обмеженнями (T extends A & B & C).
19. З яких символів можуть бути складені ідентифікатори.
20. Code convention: іменування полів, методів, класів.
21. Що таке спадкування, ключові слова implements, extends.
22. Що таке інкапсуляція, для чого призначена, як реалізувати.
23. Ключове слово final, контексти використання.
24. Конструктори, їх призначення і відміну від методів.
25. Обмеження при перекритті методу.
26. Статичні методи і поля, відміну від нестатичних.
27. Блоки ініціалізації, які бувають.
28. Порядок виклику конструкторів, блоків ініціалізації при створенні об'єкта
29. Вкладені класи, анонімні класи, написати приклад.
30. У чому відмінність вкладених класів від внутрішніх.
31. Потоки введення / виведення, класифікація, привести приклади класів.
32. InputStreamReader / OutputStreamWriter.
33. Для чого ввели дві категорії потоків - символьні і байтові.
34. Кодування, приклади кодувань, кодування за замовчуванням.
35. Кодування за замовчуванням properties файлів.
36. Клас Character. У чому відмінність між методами isDigit (int) / isDigit (char)
37. Unicode. UTF. Види, відмінності. BE / LE варіанти. BOM.
38. Документування коду, яким чином здійснюється.

39. enum. Чи можна наділити різною функціональністю елементи.
40. Строкові класи, відмінності, потокобезпечна.
41. Операції + = & &&. Типи операндів.
42. Пул рядків, для чого потрібен. Інтернування рядків. Метод intern.
43. equals. Де визначено, призначення, контракт (5 пунктів).
44. Регулярні вирази. Для чого потрібні. Написати приклад.
45. Pattern / Matcher. Чи є в класі String кошти для роботи з рег. виразами, які?
46. i18n / l10n. В чому різниця. Які засоби є для роботи з i18n.
47. Колекції. Ієрархія інтерфейсів, класів. Collections.
48. Lists. Які є, в чому відмінність.
49. Iterator / Iterable. У яких пакетах визначені, що містять.
50. Способи проходження по колекції (list / set). Написати приклади.
51. Sets. Які є, в чому відмінності.
52. Comparable / Comparator. Параметризовані? Що містять, контракт методів.
53. hashCode. Де визначено. Контакт. Зв'язок з іншими методами.
54. Яким чином використовують hashCode. Як працює hash-таблиця.
55. Зв'язок між equals і compareTo / compare.
56. Maps. Інтерфейси, класи, в чому відмінність.
57. Потоки виконання. Створення. Для чого потрібен Runnable.
58. Життєвий цикл потоку. Коли закінчує своє виконання.
59. Потоки демони, властивості, для чого потрібні, як зробити потік демоном.
60. Синхронізація. Для чого потрібна.
61. Що може бути синхронізовано.
62. Монітор синхронізації і його роль в розмежуванні доступу до синхрон. коду.
63. Методи sleep / wait. В чому різниця.
64. Фантомні пробудження.
65. Метод join.
66. Методи interrupt, interrupted, isInterrupted.
67. Як отримати поточний потік.
68. Які методи знімають блокування з монітора.
69. Exceptions. Ієрархія. Для чого потрібні.
70. Checked / unchecked виключення. В чому різниця.
71. throw / throws.
72. try / catch / finally
73. Як створити своє виключення.
74. Як викинути виняток, якщо метод не допускає секцію throws.