




Lesson 11

26.01.2026

```
public class Ex1 {  
    public static void main(String[] args) {  
        Long a = null;  
        long b = 0;  
  
        if (a == b) {  
            System.out.println("==");  
        } else {  
            System.out.println("!=");  
        }  
    }  
}
```

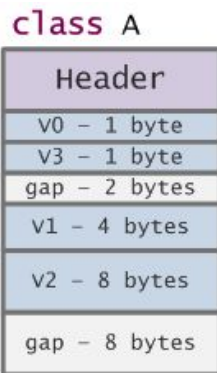
```
public class Ex2 {  
    public static void main(String[] args) {  
        System.out.println(null);  
    }  
}
```

```
public class Ex3 extends Ex3_1 {  
    public void print(int d) {  
        System.out.println("EX3");  
    }  
  
    public static void main(String[] args) {  
        Ex3 ex = new Ex3();  
        ex.print(1);  
        ex.print(2.0);  
    }  
}  
  
class Ex3_1 {  
    public void print(double d) {  
        System.out.println("EX3_1");  
    }  
}
```

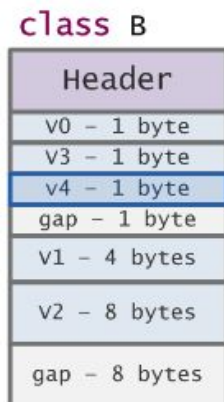


```
public class Ex5 {  
    public static void main(String[] args) {  
        TreeSet<String> set = new TreeSet<>();  
        set.add("Java");  
        set.add("The");  
        set.add("Java");  
        set.add("JavaTheBest");  
  
        for (String str : set) {  
            System.out.print(str + " ");  
        }  
        System.out.println("\n");  
    }  
}
```

```
public class A {
    byte v0;
    int v1;
    long v2;
    byte v3;
}
```



```
public class B
    extends A {
    byte v4;
}
```



Перш ніж визначати об'єм пам'яті, слід розібратися, що ж JVM зберігає для кожного об'єкта:

1. Заголовок об'єкта;
2. Пам'ять для примітивних типів;
3. Пам'ять для reference типів;
4. Зміщення/вирівнювання - по суті, це кілька байт, що не використовуються, що розміщуються після даних самого об'єкта. Це зроблено для того, щоб адреса в пам'яті завжди була кратним машинного слова, для прискорення читання з пам'яті + зменшення кількості біт для вказівника на об'єкт + імовірно для зменшення фрагментації пам'яті. Варто також відзначити, що в java розмір будь-якого об'єкта кратний 8 байтам!



32 bytes



Структура заголовка об'єкта

Кожен екземпляр класу містить заголовок. Кожен заголовок більшості JVM(Hotspot, openJVM) складається із двох машинних слів. Якщо йдеться про 32-розрядну систему, то розмір заголовка - 8 байт, якщо мова про 64-розрядну систему, то відповідно - 16 байт. Кожен заголовок може містити таку інформацію:

1. Маркувальне слово (mark word)

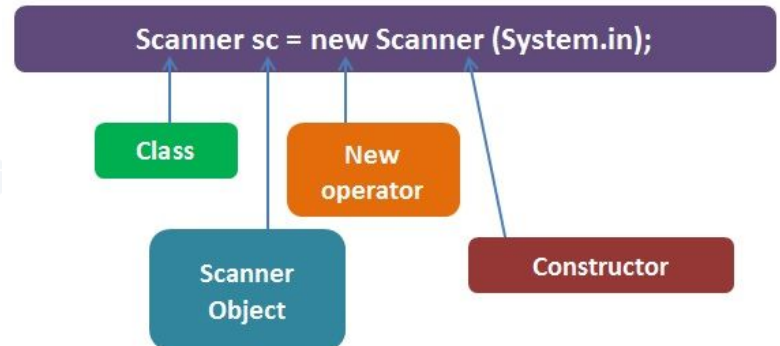
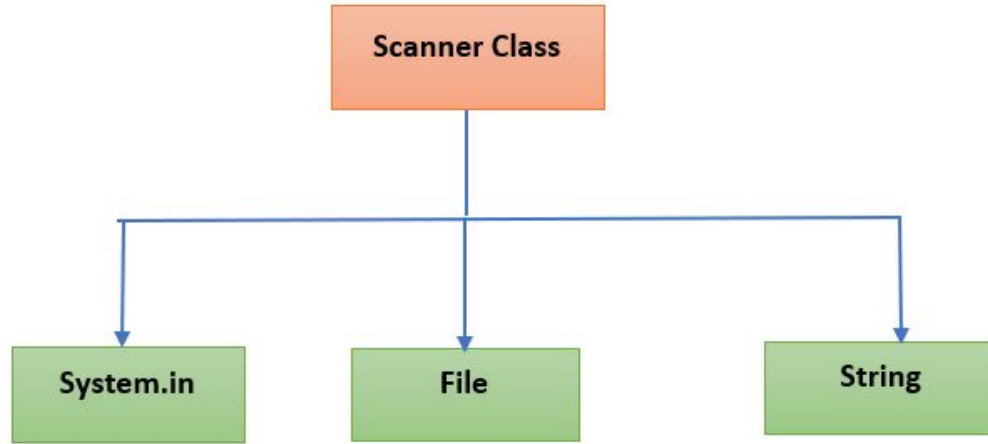
2. Hash Code – кожен об'єкт має хеш-код. За замовчуванням результат виклику методу `Object.hashCode()` поверне адресу об'єкта в пам'яті, проте деякі збирачі сміття можуть переміщувати об'єкти в пам'яті, але хеш код завжди залишається одним і тим же, оскільки місце в заголовку об'єкта може бути використане для зберігання оригінального значення хеш-коду.

3. Garbage Collection Information - кожен java об'єкт містить інформацію, потрібну для системи управління пам'яттю. Найчастіше це один або два біти-прапори, але це може бути, наприклад, якась комбінація бітів для зберігання кількості посилань на об'єкт.

4. Type Information Block Pointer – містить інформацію про тип об'єкта. Цей блок включає інформацію про таблицю віртуальних методів, покажчик на об'єкт, який представляє тип і покажчики деякі додаткові структури, для ефективніших викликів інтерфейсів і динамічної перевірки типів.

5. Lock – кожен об'єкт містить інформацію про стан блокування.

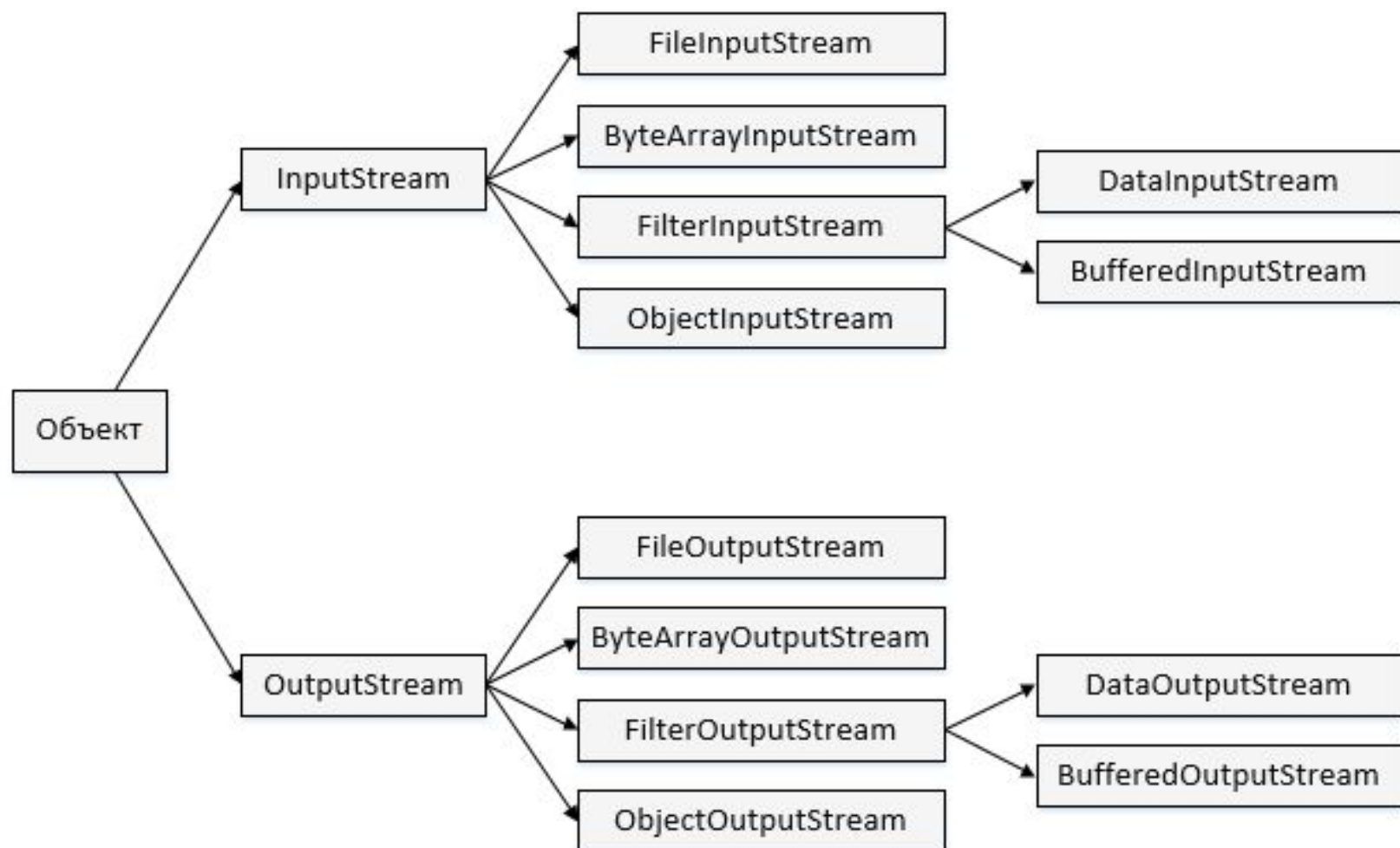
Different input sources





How to take input from user in Java using Scanner?

Method	Description
int nextInt()	Used for scanning the next token as an integer
float nextFloat()	Used for scanning the next token as a float
double nextDouble()	Used for scanning the next token as a double
byte nextByte()	Used for scanning the next token as a byte
String nextLine()	Advances this scanner past the current line
boolean nextBoolean()	Used for scanning the next token into a boolean value
long nextLong()	Used for scanning the next token as a long
short nextShort()	Used for scanning the next token as a Short
BigInteger nextBigInteger()	Used for scanning the next token as a BigInteger
BigDecimal nextBigDecimal()	Used for scanning the next token as a BigDecimal





Поток FileInputStream – читання з файлу

```
InputStream a = new FileInputStream("D:/myprogramm/java/test");
```

```
File a = new File("D:/myprogramm/java/test");  
InputStream a = new FileInputStream(a);
```



Поток `FileOutputStream` – створення та запис у файл

```
OutputStream a = new FileOutputStream("D:/myprogramm/java/test")
```

```
File a = new File("D:/myprogramm/java/test");  
OutputStream a = new FileOutputStream(a);
```



```
public String getName()
```

```
public String getParent()
```

```
public boolean canRead()
```

```
public boolean canWrite()
```

```
public boolean exists()
```

```
public boolean isDirectory()
```



```
public boolean isDirectory()
```

```
public long lastModified()
```

```
public boolean createNewFile() throws IOException
```

```
public boolean delete()
```

```
public void deleteOnExit()
```

```
public String[] list()
```

Lombok



Spice up your JAVA

Setting up Lombok with IntelliJ

@NonNull

or: How I learned to stop worrying and love the NullPointerException.

@Cleanup

Automatic resource management: Call your `close()` methods safely with no hassle.

@Getter/@Setter

Never write `public int getFoo() {return foo;}` again.

@ToString

No need to start a debugger to see your fields: Just let lombok generate a `toString` for you!

@EqualsAndHashCode

Equality made easy: Generates `hashCode` and `equals` implementations from the fields of your object..

@NoArgsConstructor, @RequiredArgsConstructor and @AllArgsConstructor

Constructors made to order: Generates constructors that take no arguments, one argument per final / non-nullfield, or one argument for every field.

@Data

All together now: A shortcut for `@ToString`, `@EqualsAndHashCode`, `@Getter` on all fields, and `@Setter` on all non-final fields, and `@RequiredArgsConstructor`!

@Value

Immutable classes made very easy.

@Builder

... and Bob's your uncle: No-hassle fancy-pants APIs for object creation!

@SneakyThrows

To boldly throw checked exceptions where no one has thrown them before!

@Synchronized

`synchronized` done right: Don't expose your locks.

@With

Immutable 'setters' - methods that create a clone but with one changed field.

@Getter(lazy=true)

Laziness is a virtue!

@Log

Captain's Log, stardate 24435.7: "What was that line again?"



Lombok Annotations	Converted File
<pre>import lombok.Getter; import lombok.Setter; @Getter public class Book { private String id; private String title; private double price; @Setter private String author; @Setter private boolean available; }</pre>	<pre>Book.java Book author available id price title getAuthor() : String getId() : String getPrice() : double getTitle() : String isAvailable() : boolean setAuthor(String) : void setAvailable(boolean) : void</pre>
©javatechonline.com	

