Magasszintű programozás 2 – elméleti rész lehetséges kérdései

2. számonkérés

11. Mi az az I/O Stream?

- An *I/O Stream* represents an input source or an output destination.
- A stream can represent many different kinds of sources and destinations, including disk files, devices, other programs, and memory arrays.
- Streams support many different kinds of data, including simple bytes, primitive data types, localized characters, and objects. Some streams simply pass on data; others manipulate and transform the data in useful ways.

12. Mi az a byte stream?

- Programs use *byte streams* to perform input and output of 8-bit bytes.
- All byte stream classes are descended from InputStream and OutputStream.
- There are many byte stream classes. We will check the **file I/O byte streams**, FileInputStream and FileOutputStream.
- Other kinds of byte streams are used in much the same way; they differ mainly in the way they are constructed.

13. Mi az a character stream?

- The Java platform stores character values using **Unicode** conventions. Character stream I/O automatically translates this internal format to and from the local character set. In Western locales, the local character set is usually an 8-bit superset of ASCII.

14. Mi az a buffered stream?

- Previous examples were *unbuffered* I/O. This means each read or write request is handled directly by the underlying OS. This can make a program much less efficient, since each such request often triggers disk

access, network activity, or some other operation that is relatively expensive.

- To reduce this kind of overhead, the Java platform implements ***buffered* I/O streams**. Buffered input streams read data from a memory area known as a *buffer*; **the native input API is called only when the buffer is empty**. Similarly, buffered output streams write data to a buffer, and **the native output API is called only when the buffer is full**.

15. Scanning and Formatting

- Programming I/O often involves translating to and from the **neatly formatted data humans like to work with**. To assist you with these chores, the Java platform provides two APIs.
- The scanner API breaks input into **individual tokens** associated with bits of data.
- The formatting API assembles data into **nicely formatted, human-readable form.**

16. Mi a 3 standard stream a Java-ban? Melyik mire való?

- The Java platform supports three Standard Streams:
    - *Standard Input*, accessed through System.in;
    - *Standard Output*, accessed through System.out; and
    - *Standard Error*, accessed through System.err.

17. Mi a data stream?

- Data streams support binary I/O of primitive data type values (boolean, char, byte, short, int, long, float, and double) as well as String values. All data streams implement either the DataInput interface or the DataOutput interface.
- The most widely-used implementations of these interfaces, DataInputStream and DataOutputStream.

18. Mi az object stream?

- Object streams support I/O of objects. Most, but not all, standard classes support serialization of their objects. Those that do implement the marker interface Serializable.

21. Mit csinál a Javadoc eszköz?

- *javadoc* tool
- create a standard documentation of Java code in HTML file format.
- Java officially uses this tool to create its own library API documentation.
- There are certain **comments** that we want to show up in the documentation. The style of writing these comments in the source code begins with /** and ends with */.
- Any text written within these two markers are designated as **documentation comments**. This is similar to traditional multiline comments used in Java. The text within these two markers also can **span multiple lines**.
-

22. Soroljon fel 4 tag-et a Javadoc eszközhöz, és magyarázza, hogy mit csinálnak.

@author name-text: This tag is used to insert the author's name into the generated docs, designated by *name-text* .

{@code text}: This is equivalent to writing <code>{@literal}</code>. This means that the text embraced within the tags will be rendered in a code font.

@deprecated deprecated-text: This tag is used to designate a text as deprecated, meaning that the text will be rendered with a bold warning of "Deprecated." The program element is, however, deprecated with the *@Deprecated* annotation.

@see reference: This tag adds a "See Also" heading with the text entry to links to the reference.

23. Javadoc eszköz használata esetén a fő leírást hova írja? Milyen elemekhez adhat meg dokumentációt?

- **Documentation comm**ents are recognized only when placed immediately **before module, package, class, interface, constructor, method, enum member, or field decl**arations. Documentation comments placed in the body of a method are ignored. Only one documentation comment per declaration statement is recognized.

31. Mire való a JUnit?

- A JUnit egy népszerű keretrendszer a Java nyelvhez, amely lehetővé teszi az egységtesztek könnyű írását és végrehajtását. Használata hatékony módja a Unit tesztek készítésének és a kód minőségének javításának.

32. Soroljon fel a JUnit eszközben használatos annotációkból ötöt és magyarázza mire valók!

- @Test
    - Denotes that **a method is a test method**. Unlike JUnit 4's @Test annotation, this annotation does not declare any attributes, since test extensions in JUnit Jupiter operate based on their own dedicated annotations. Such methods are inherited unless they are overridden.
- @ParameterizedTest
    - Denotes that **a method is a parameterized test**. Such methods are inherited unless they are overridden.
- @RepeatedTest
    - Denotes that **a method is a test template for a repeated test**. Such methods are inherited unless they are overridden.
- @TestFactory
    - Denotes that **a method is a test factory for dynamic tests**. Such methods are inherited unless they are overridden.

- @TestTemplate

- Denotes that **a method is a template for test cases designed to be invoked multiple times** depending on the number of invocation contexts returned by the registered providers. Such methods are inherited unless they are overridden.

33. A JUnit eszköz assertation-jai mire valók? Soroljon fel belőlük 10-et és magyarázza őket!

- assertEquals, assertNotEquals, assertFalse, assertTrue, assertThrows, assertDoesNotThrows, AssertTimeout, assertSame, assertNotSame, assertNull, assertInstanceOf, assertIterableEquals, assertArrayEquals, assertAll

41. A kollekciók aggregáló műveletei esetén a pipeline-nak milyen részei vannak? Melyik mire való?

A *pipeline* is a sequence of aggregate operations.

- A pipeline contains the following components:
    1. A **source**: This could be a **collection**, an **array**, a **generator function**, or an **I/O channel**.
    2. Zero or more ***intermediate operations***. An intermediate operation, such as **filter**, produces a new stream.
    3. A ***terminal operation***. A terminal operation, such as **forEach**, produces a **non-stream result**, such as a **primitive value** (like a double value), a **collection**, or in the case of forEach, **no value at all**.

42. Hogyan működik a reduce művelet a kollekciók aggregáló műveletei esetén?

The Stream.reduce method is a general-purpose reduction operation.

.reduce(0, (a, b) -> a + b); = .sum()

43. Hogyan működik a collect művelet a kollekciók aggregáló műveletei esetén?

- The collect method modifies, or mutates, an existing value.
- .collect(Collectors.toList());

- .stream().collect(Collectors.groupingBy(Person::getGender, Collectors.mapping(Person::getName, Collectors.toList())));

44. Mutassa be a kollekciók aggregáló műveleteiből a groupingBy és reducing műveleteket!

.collect(Collectors.groupingBy(Person::getGender, Collectors.reducing(0, Person::getAge, Integer::sum)));

51. A JAR eszköz mire való, milyen műveleteket lehet vele elvégezni?

- The Java™ Archive (JAR) file format enables you to bundle multiple files into a single archive file. Typically a JAR file contains the class files and auxiliary resources.

Common JAR file operations

- To create a JAR file:
  - jar cf *jar-file input-file(s)*
- To view the contents of a JAR file
  - jar tf *jar-file*
- To extract the contents of a JAR file
  - jar xf *jar-file*
- To extract specific files from a JAR file
  - jar xf *jar-file archived-file(s)*
- To run an application packaged as a JAR file (requires the Main-class manifest header)
  - java -jar *app.jar*

52. A JAR file „manifest”-jét mutassa be!

- When you create a JAR file, it automatically receives a default manifest file. There can be only one manifest file in an archive, and it always has the pathname
  - META-INF/MANIFEST.MF

- When you create a JAR file, the default manifest file simply contains the following:

Manifest-Version: 1.0

61. Mi az a Java modul?

- What Is a Module?
    - Modularity adds a higher level of aggregation above packages. The key new language element is the **module**—a uniquely named, reusable group of related packages, as well as resources (such as images and XML files) and a **module descriptor** specifying
        - the module's **name**
        - the module's **dependencies** (that is, other modules this module depends on)
        - the packages it explicitly makes available to other modules (all other packages in the module are **implicitly unavailable** to other modules)
        - the **services it offers**
        - the **services it consumes**
        - to what other modules it allows **reflection**

62. A Java modul module-info.java állományában mit jelentenek az exports, exports … to, requires, uses direktívák?

- **requires.** A requires module directive specifies that this module depends on another module—this relationship is called a **module dependency**. Each module must explicitly state its dependencies. When module A requires module B, module A is said to **read** module B and module B is **read by** module A. To specify a dependency on another module, use requires, as in:

requires **modulename**;

- **exports and exports…to.** An exports module directive specifies one of the module's packages whose public types (and their nested public and protected types) should be accessible to code in all other

modules. An exports…to directive enables you to specify in a comma-separated list precisely which module's or modules' code can access the exported package—this is known as a **qualified** export.

- **uses.** A uses module directive specifies a service used by this module—making the module a service consumer. A **service** is an object of a class that implements the interface or extends the abstract class specified in the uses directive.

71. Mi az a Project Lombok?

- Project Lombok is a java library that automatically plugs into your editor and build tools, spicing up your java.
  Never write another getter or equals method again, with one annotation your class has a fully featured builder, Automate your logging variables, and much more.

72. Project Lombok-kal hogyan ad meg gettert, settert?

```
import lombok.AccessLevel;
import lombok.Getter;
import lombok.Setter;

public class GetterSetterExample {
  @Getter @Setter private int age = 10;

  @Setter(AccessLevel.PROTECTED)
    private String name;

  @Override public String toString() {
   return String.format("%s (age: %d)", name, age);
  }
}
```

73. Project Lombok-kal hogyan ad meg toString-et?

```
import lombok.ToString;

@ToString
public class ToStringExample {
  private static final int STATIC_VAR = 10;
  private String name;
  private Shape shape = new Square(5, 10);
  private String[] tags;
  @ToString.Exclude private int id;

  public String getName() {
    return this.name;
  }

  @ToString(callSuper=true, includeFieldNames=true)
  public static class Square extends Shape {
    private final int width, height;

    public Square(int width, int height) {
      this.width = width;
      this.height = height;
    }
  }
}
```

74. Project Lombok-kal hogyan ad meg equals-t és HashCode-ot?

```
  import lombok.EqualsAndHashCode;

@EqualsAndHashCode
public class EqualsAndHashCodeExample {
  private transient int transientVar = 10;
  private String name;
  private double score;
  @EqualsAndHashCode.Exclude private Shape shape = new Square(5, 10);
  private String[] tags;
  @EqualsAndHashCode.Exclude private int id;
```

```
  public String getName() {
    return this.name;
  }

  @EqualsAndHashCode(callSuper=true)
  public static class Square extends Shape {
    private final int width, height;

    public Square(int width, int height) {
      this.width = width;
      this.height = height;
    }
  }
}
```