

№ 2 -3 Основы создания интерфейса

Задание

- 1) Разработайте интерфейс калькулятора с функционалом:
 - ✓ Содержит кнопки с цифрами от 0 до 9
 - ✓ Бинарные операции: +, -, /, *
 - ✓ Унарные операции \pm , $\sqrt{}$, cos, sin, еще одну операцию на выбор
 - ✓ =, . (точка), π
- 2) Сконструируйте «растягивающийся» пользовательский интерфейс (UI), который будет работать в портретном и ландшафтном режимах. Все кнопки и расстояния между ними должны быть выровнены.
- 3) Функции калькулятора должны быть вынесены в отдельный класс (интерфейс) Operations, типы операций – заданы перечислением (унарные, бинарные, равно, константные), а набор операций представлен в виде словаря, так чтобы добавление новых функций к калькулятору было максимально удобно и легко.
- 4) Используйте лямбда выражения (Java 8)
- 5) Для отладки примените всплывающие сообщения и логгирование.
- 6) Соблюдать java code convention.

Логгирование.

Логгирование является средством отладки. Существуют разные фреймворки для этого. Вам предлагается взять Java.util.logging. Данный фреймворк включен в стандарт и поставляется вместе с JDK, поэтому ничего дополнительно скачивать и подключать вам не надо. В нем существуют следующие уровни логгирования по возрастанию: FINEST, FINER, FINE, CONFIG, INFO, WARNING, SEVERE, а так же ALL и OFF, включающий и отключающий все уровни соответственно. Логгер создается вызовом одного из статических методов класса java.util.logging.Logger:

```
private static final Logger LOG = Logger.getLogger(Main.class.getName());
```

Методы логгера могут принимать в качестве аргументов строковые сообщения, шаблоны сообщений, исключения, ресурсы локализованных текстовок сообщений, а также, начиная с Java 8, поставщиков строковых сообщений.

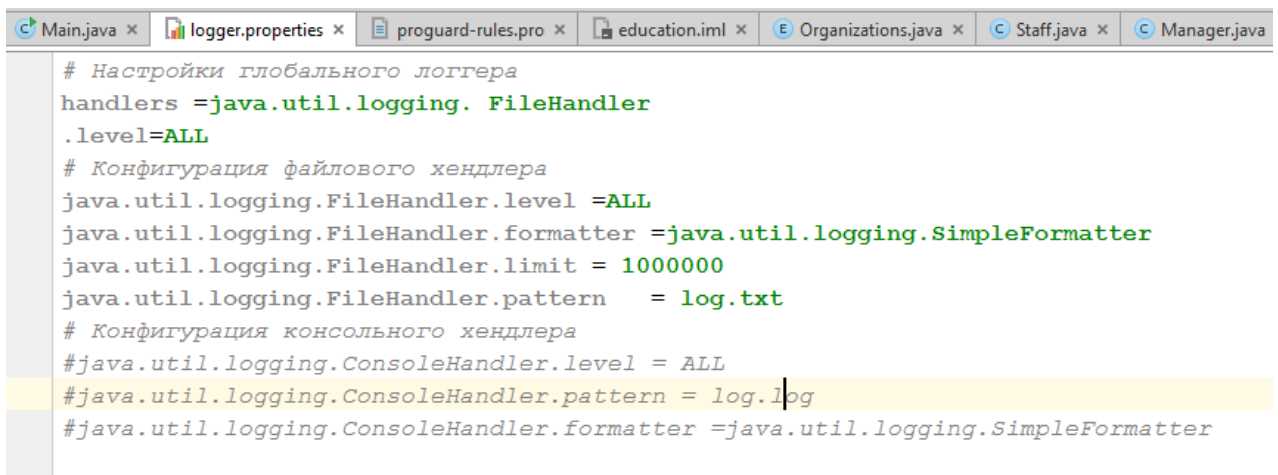
Выделяется две группы методов: название которых соответствует уровню логгирования и методы `log`, `loggpr`, `logrb`, принимающие уровень логгирования в качестве параметра с типом `Level`. Первая группа содержит методы двух типов: принимающих строковое сообщение или поставщика строковых сообщений:

```
LOG.info("Starting program_____");
LOG.info(dataBase.toString());
LOG.info("Sum Rangers = " + dataBaseManger.sumRanges(dataBase));
//
```

Вторая группа методов использует уровень логгера, а также может выводить имя класса, имя метода и т.п:

```
LOG.log(Level.INFO, "Main" , "Start programm");
```

По умолчанию вывод работает на консоль, однако можно задать конфигурацию в файле свойств. Для задания способа вывода сообщений необходимо для логгера указать какие хендлеры он будет использовать. Существует следующие классы хендлеров: `FileHandler`, `ConsoleHandler`, `StreamHandler`, `SocketHandler`, `MemoryHandler`. Особенностью является то, что настройки хендлеров задаются в целом для всего класса, а не для конкретного экземпляра. Вот пример конфигурационного файла:

The screenshot shows an IDE window titled 'logger.properties'. The file contains configuration for a logger. It sets the handlers to 'java.util.logging.FileHandler' and the level to 'ALL'. It also configures the FileHandler with a formatter of 'java.util.logging.SimpleFormatter', a limit of 1000000, and a pattern of 'log.txt'. Additionally, it configures the ConsoleHandler with the same level and formatter, and a pattern of 'log.log'.

```
# Настройки глобального логгера
handlers =java.util.logging.FileHandler
.level=ALL
# Конфигурация файлового хендлера
java.util.logging.FileHandler.level =ALL
java.util.logging.FileHandler.formatter =java.util.logging.SimpleFormatter
java.util.logging.FileHandler.limit = 1000000
java.util.logging.FileHandler.pattern = log.txt
# Конфигурация консольного хендлера
#java.util.logging.ConsoleHandler.level = ALL
#java.util.logging.ConsoleHandler.pattern = log.log
#java.util.logging.ConsoleHandler.formatter =java.util.logging.SimpleFormatter
```

Для применения данной конфигурации нужно передать параметр `-Djava.util.logging.config.file = <путь до файла>`, либо при старте приложения выполнить код:

```
static{
    try {
        LogManager.getLogManager().readConfiguration(new
FileInputStream("logger.properties"));
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```