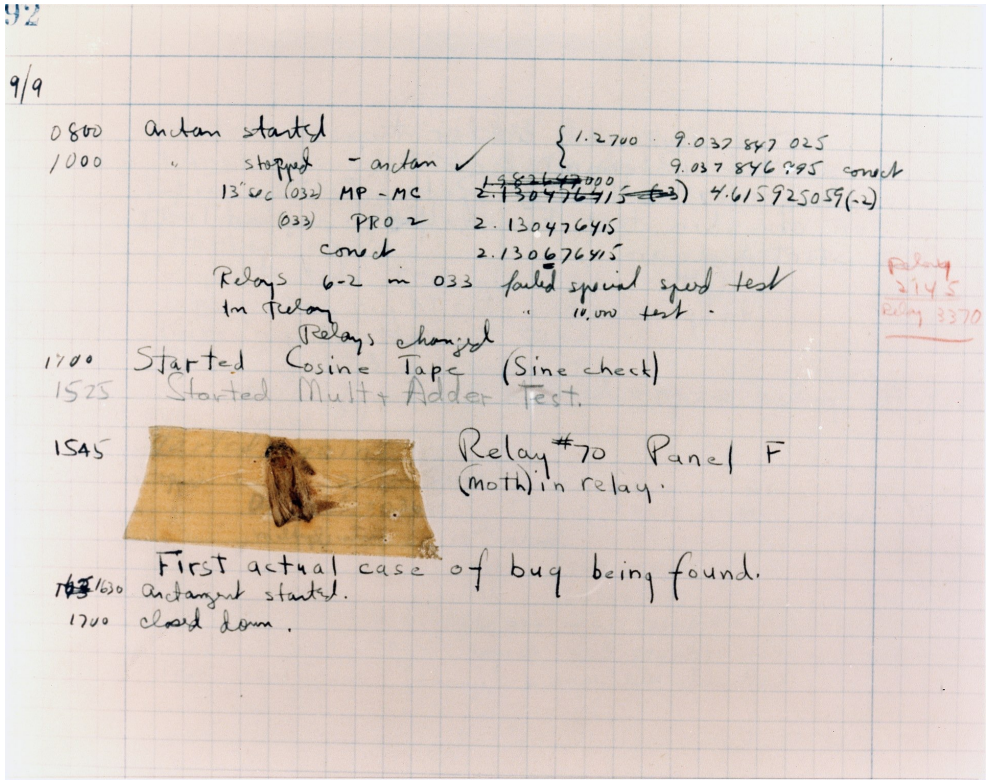
Java Debugging

‘Debug’ – этимология

- Произошло от слова "Bugs" (насекомые)

Наиболее популярная версия:

- Компьютер Mark II вышел из строя в 1940-х из-за моли в реле (на изображении лог-журнал с записью: “*moth in relay*”)
- Во время починки компьютера Grace Hopper (один из разработчиков) назвала этот процесс как ‘debugging the system’.
- Debug – отладка программы



Source

Базовые концепции Отладки

- Отладка – неотъемлемая часть разработки ПО, которая включает в себя поиск и устранение ошибок или проблем в программе
- Ошибки выражаются в виде неправильного поведения, сбоев или неожиданных результатов, а отладка помогает найти и устранить их
- Отладка — это систематический подход к поиску и устранению ошибок ПО
  - Выявление, изоляция, исправление ошибок

Базовые концепции Отладки

- Отладка состоит из нескольких шагов:
  1. Воспроизведение проблемы: нужно понять **условия**, которые приводят к ошибке.
  2. Обнаружение ошибки: определение **конкретного кода или логики**, где возникает ошибка.
  3. Анализ ошибки: **понимание основной причины** и влияния ошибки на поведение программы.
  4. Исправление ошибки: вносим **правки** в код для восстановления функциональности.
  5. Тестирование: проверка того, что ошибка была успешно исправлена и программа работает должным образом.

Базовые концепции Отладки

- Почему важно подходить к отладке системно:
  - Отладка может быть сложным и трудоемким процессом, особенно в крупномасштабных проектах ПО.
  - Системный подход к отладке помогает сократить время и усилия, необходимые для выявления и исправления ошибок.
  - Подходя к отладке систематически, разработчики могут повысить эффективность и точность решения проблем.

Фреймворки для отладки

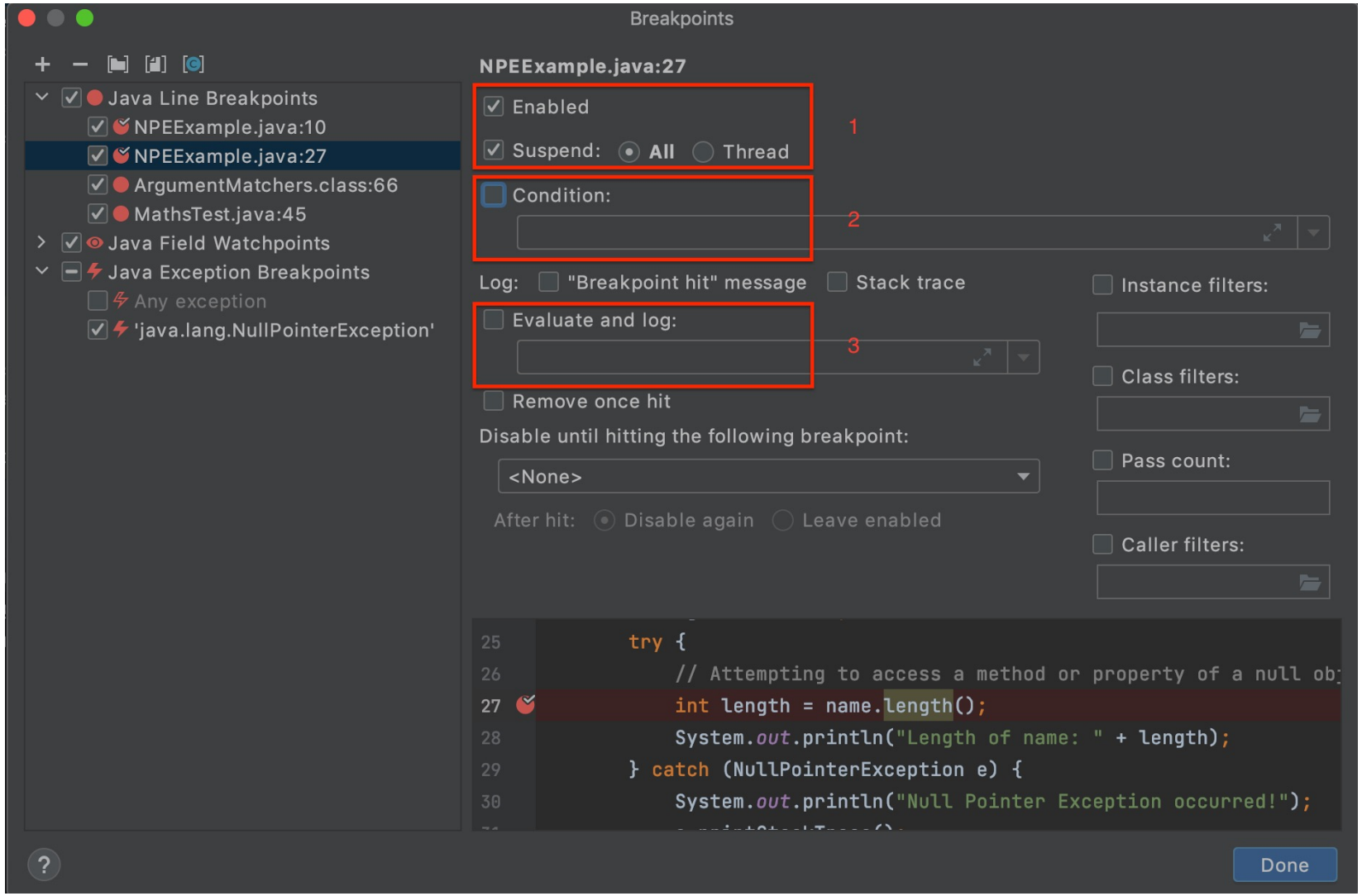
- Логгирование: Log4J, SLF4J (см. предыдущую презентацию)
- В зависимости от степени «серьёзности», логи делят на несколько уровней: TRACE, DEBUG, INFO, WARN, ERROR, FATAL/CRITICAL.
  - **TRACE** – слишком большая детализация информации, вплоть до вызова каждого метода. Помогают в диагностике сложных проблем, которые сложно воспроизвести
  - **DEBUG** – отладочная информация
  - **INFO** – обычное поведение приложения, «дежурная» информация
  - **WARN** – потенциальные проблемы, неожиданное поведение
  - **ERROR** – ошибка, влияет на поведение приложения, нужно срочно обратить внимание, сбой работы приложения
- Профилировщики – позволяют понять, сколько по времени выполняется кусок кода, etc.

Логгирование

- Логгирование играет решающую роль в отладке Java-программы
- Оно помогает собрать ценную информацию о выполнении программы
- Различные уровни логгирования в сочетании с верным размещением logging statements помогают получить представление:
  - о поведении программы, отследить выполнение кода и выявить потенциальные проблемы.
  - Это помогает упростить процесс отладки и повысить общую надежность и удобство обслуживания приложений Java.

Инструменты для отладки. IntelliJ IDEA

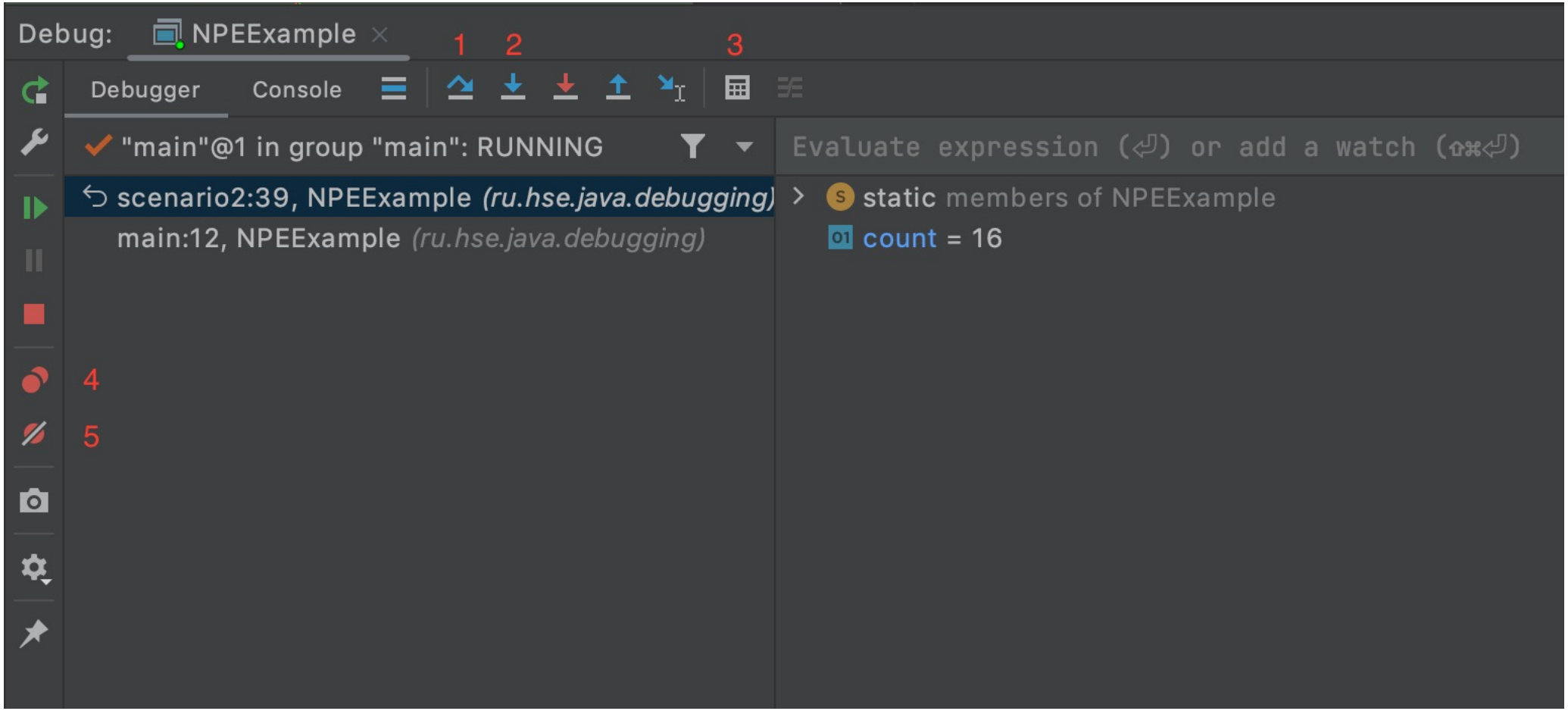
- В отладке «здесь и сейчас» помогает **I**ntegrated **D**evelopment **E**nvironments (**IDE**s)
- IntelliJ IDEA обладает более совершенным аппаратом для отладки:
  - **Breakpoints** – точки останова. Разработчик ставит точку останова в определённых строках кода, программа переключается в режим отладки.
  - **Stepping through code** (line by line) – построчное выполнение программы
  - **Variable inspection** – «проверка переменных». Можно в рантайме проверять значения переменных, выражений и т.д.
  - **Call stack** – IDE отображает стек вызовов методов (трассировку), который ведёт к последовательности вызова методов, которые были выполнены до текущей точки выполнения. *Это помогает понять ход выполнения программы и определить источники ошибок.*
  - **Watch expressions** – можно вынести перемененную, выражение в окно Debugger и узнать её значение/следить за обновлением значения.
  - https://www.jetbrains.com/help/idea/debugging-code.html



1: стандартные настройки точки останова

2: точка останова отрабатывает, только если выполняется условие

3: если выставить поле *suspend* в false, то можно залоггировать значение переменной



1: идёт на следующую строчку кода, не «проваливается» в вызовы вложенных методов, конструкторов
2: пойдёт глубже, в код вызываемых методов/конструкторов
3: окно для выполнения выражения в данном контексте метода (видны только переменные внутри контекста, фигурных скобок метода)
4: покажет все точки останова
5: отключит все точки останова