



**Universidade Federal do Ceará**  
**Centro de Ciências/Departamento de Computação**  
**Código da Disciplina:** CK0236  
**Professor:** Ismayle de Sousa Santos

**Aula 10**

# **Técnica de Programação II**

## **Depuração de Software**

---



**qpg4p5x**



**ismaylesantos@great.ufc.br**



**@IsmayleSantos**

# Erro x Defeito x Falha

- **Erro**
  - Uma ação humana que produz um resultado incorreto
    - E.g.: Colocar (+) ao invés de (-)
- **Defeito**
  - Uma imperfeição ou deficiência em um produto de trabalho onde este não atende aos seus requisitos ou especificações e precisa ser reparado e substituído [IEEE 1044-2009]
  - Falta é um defeito no código
    - E.g.: Está somando ao invés de subtrair
- **Falha**
  - É o comportamento externo incorreto do sistema. Acontece quando a falta é executada
  - E.g.: O sistema exibindo um valor incorreto

# Erro x Defeito x Falha



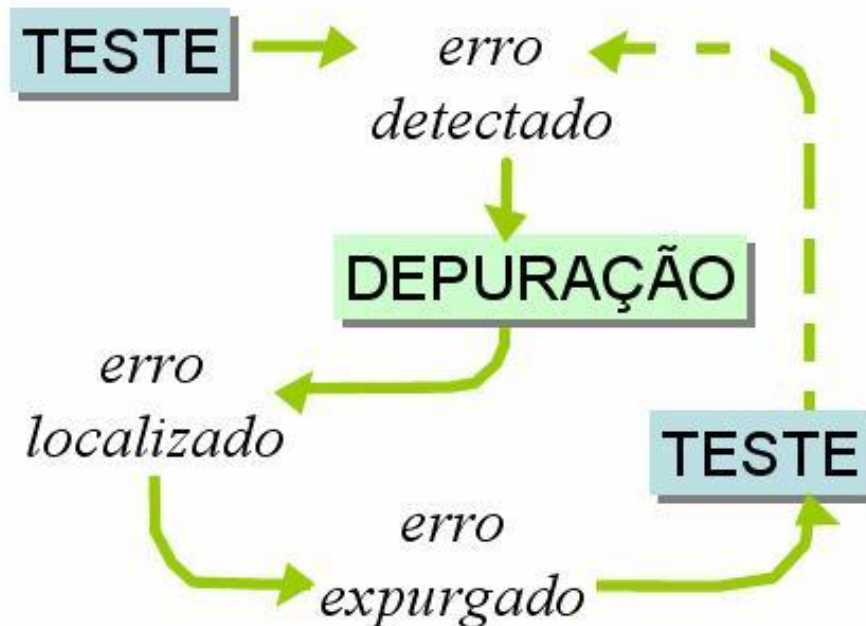
# O que é Depuração?

- É o processo de identificar a causa raiz de um defeito e corrigir ela
- Em alguns projetos ela pode ocupar até 50% do total de tempo de desenvolvimento



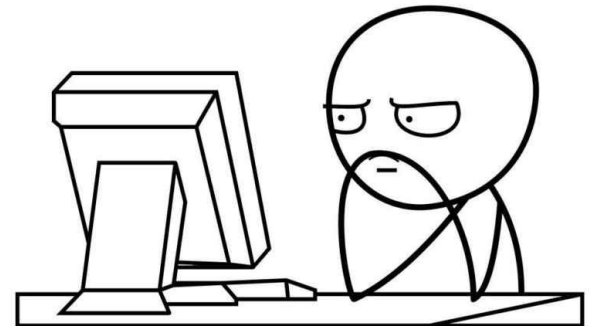
# Teste e Depuração?

- **Teste**
  - Testes podem demonstrar falhas que são causadas por defeito
- **Depuração**
  - É o processo de procurar, analisar e remover as causas e falhas no software



# Esforço na Depuração

- Pode variar bastante
- Como você depura um software para encontrar um bug?



# Oportunidades da Depuração

- A partir da depuração você pode
  - Aprender mais sobre o programa que você está desenvolvendo
  - Aprender sobre os tipos de erros que você comete
    - Como você pode encontrar esse tipo de erro mais rapidamente? Como evitá-los?
  - Analisar a qualidade do código
    - Ele está fácil de ler? O que poderia ser melhor?
  - Aprender como corrigir defeitos

# Como não Depurar

- Algumas abordagens comuns, mas ineficientes
  - Encontre o defeito por achismo
    - Usando *prints* em várias partes
    - Tentando alterar partes do programa ou comentar até encontrar o local do defeito
  - Corrija o defeito com a correção mais óbvia
    - Correções só do defeito em questão, sem corrigir o real problema no software todo



# Encontrando um Defeito

- Lembrando que encontrar um defeito e entender ele usualmente corresponde à 90% do esforço
- Passos (método científico)
  1. Reunir **dados** através de experimentos repetíveis
  2. Formule uma **hipótese** que explica os dados
  3. Projete um experimento para aceitar ou não a hipótese
  4. Aceite ou Rejeite a hipótese
  5. Repita o processo enquanto for necessário



# **Passos para depuração**

- 1. Estabilize o defeito (repetibilidade do defeito)**
- 2. Localize a origem do defeito**
  - a. Reúna dados para reproduzir o defeito**
  - b. Analise os dados e formule uma hipótese sobre o defeito**
  - c. Determine como aceitar ou rejeitar a hipótese**
  - d. Aceite ou rejeite a hipótese seguindo o passo 2(c)**
- 3. Corrija o defeito**
- 4. Teste a correção**
- 5. Procure por erros similares**

# Exemplo

- Suponha um programa que imprime nome dos projetos, nome dos funcionários e valores do imposto de renda em ordem alfabética
- Suponha que ele está imprimindo o seguinte:

Format, Gary	\$ 1000,00
Modula, Mildred	\$ 1500,00
Many-Loop, Mavis	\$ 2000,00
National, Wendy	\$ 4000,00

# Exemplo

- Suponha um programa que imprime nome dos projetos, nome dos funcionários e valores do imposto de renda em ordem alfabética
- Suponha que ele está imprimindo o seguinte:

Format, Gary	\$ 1000,00
Modula, Mildred	\$ 1500,00
Many-Loop, Mavis	\$ 2000,00
National, Wendy	\$ 4000,00

} Fora de ordem

# 1. Estabilize o defeito [Exemplo]

- É preciso verificar se a ocorrência do defeito é previsível
  - Se a ocorrência não é previsível (i.e., intermitente), pode ser um erro de inicialização, atualização de ponteiros
- Suponha que ele está imprimindo o seguinte:

Format, Gary	\$ 1000,00
Modula, Mildred	\$ 1500,00
Many-Loop, Mavis	\$ 2000,00
National, Wendy	\$ 4000,00

Fora de ordem

# 1. Estabilize o defeito [Exemplo]

- Você executa de novo (segunda vez) e a saída fica correta



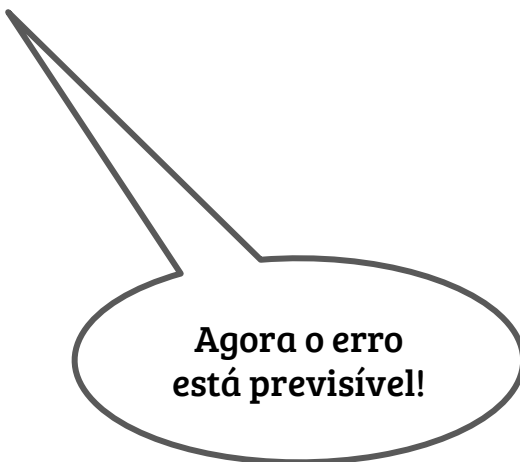
Format, Gary	\$ 1000,00
Many-Loop, Mavis	\$ 2000,00
Modula, Mildred	\$ 1500,00
National, Wendy	\$ 4000,00

- Porém, você adiciona um novo funcionário (Fruit-Loop, Ana \$ 2500,00) e novamente a saída fica errada com a nova entrada aparecendo antes de 'Format, Gary \$ 1000,00'

# 1. Estabilize o defeito [Exemplo]

- Então você cria uma hipótese:
  - “O problema tem haver com a entrada de um único funcionário”
  - Você executa novamente, e confirma sua hipótese com a saída correta da impressão na segunda execução

Format, Gary	\$ 1000,00
Fruit-Loop, Ana	\$ 2500,00
Many-Loop, Mavis	\$ 2000,00
Modula, Mildred	\$ 1500,00
National, Wendy	\$ 4000,00



Agora o erro  
está previsível!

## 2. Localize a Origem do Defeito [Exemplo]

- Após analisar os dados, você cria uma hipótese da origem do defeito
  - “Pode ser um *“off-by-one bug”* (e.g., no código, o loop que processa os funcionários não processou o último do array)
  - Você examina o código e não parece ser isso, então você faz um outro teste adicionando ‘Hardcase, Henry \$ 500,00’ e a saída fica correta!!

Format, Gary	\$ 1000,00
Fruit-Loop, Ana	\$ 2500,00
Hardcase, Henry	\$ 500,00
Many-Loop, Mavis	\$ 2000,00
Modula, Mildred	\$ 1500,00
National, Wendy	\$ 4000,00

**Hipótese rejeitada!**  
Não é um bug da  
entrada de 1 novo  
funcionário



## 2. Localize a Origem do Defeito [Exemplo]

- Examinando novamente os testes, você percebe que em ambos os casos, os nomes tinham hífen
- Então surge uma nova hipótese:
  - “O problema é com nomes que possuem hífen”
- Espera!
  - mas isso só estava acontecendo na primeira vez que você inseriu o funcionário



## 2. Localize a Origem do Defeito [Exemplo]

- Então você reúne mais dados olhando o código e observa que são usados 2 métodos de ordenação diferentes
  - M1: ao inserir o funcionário
  - M2: quando o dado é salvo
- Olhando mais de perto, você nota que no M1 que não é usado para ordenar os dados completamente, ele apenas colocar o dado na posição aproximada para acelerar o processamento M2
  - Ou seja, o problema é que os dados foram impressos antes de serem ordenados
- Última hipótese: “nomes com pontuação não são ordenadas corretamente até serem salvas”

# Tipos de Erros

- **Sintáticos**
    - Quando a sintaxe da linguagem não está sendo respeitada
    - Exemplo: Falta de parênteses
  - **Semânticos**
    - Uso indevido de declarações do sistema
    - Exemplo: String nome = 5
  - **Lógicos**
    - Quando a especificação não é seguida
    - Origem do defeito, que por sua vez pode ocasionar na falha
    - Exemplo: Usar “+” ao invés de “-”
-

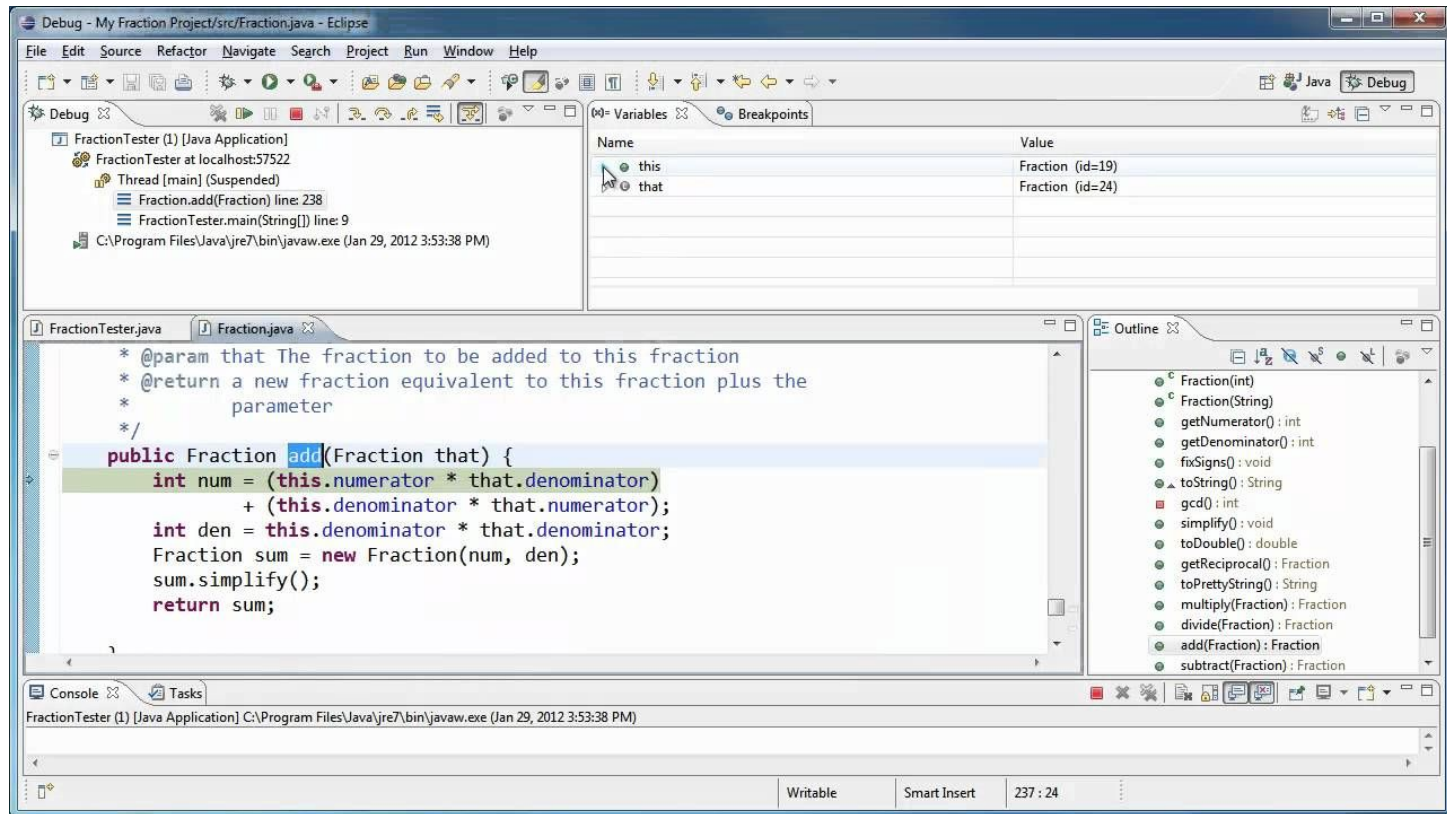
# Dicas para encontrar os defeitos

- Use todos os dados disponíveis para montar sua hipótese
  - Refinar a hipóteses junto com os dados sendo obtidos
- Refinar casos de testes que produzem a falha
  - Quanto mais específico o caso de teste, mas ficará claro quais parâmetros estão relacionados ao defeito
- Utilize testes unitários para testar as unidades isoladamente
  - É mais fácil encontrar defeitos em pequenos fragmentos de código



# Dicas para encontrar os defeitos

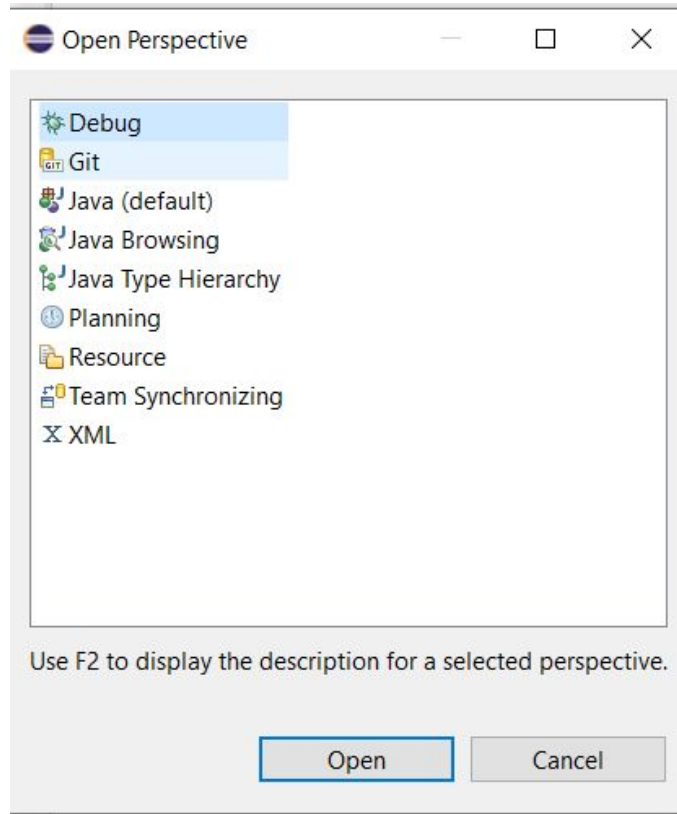
- Use ferramentas disponíveis



Eclipse Debugger

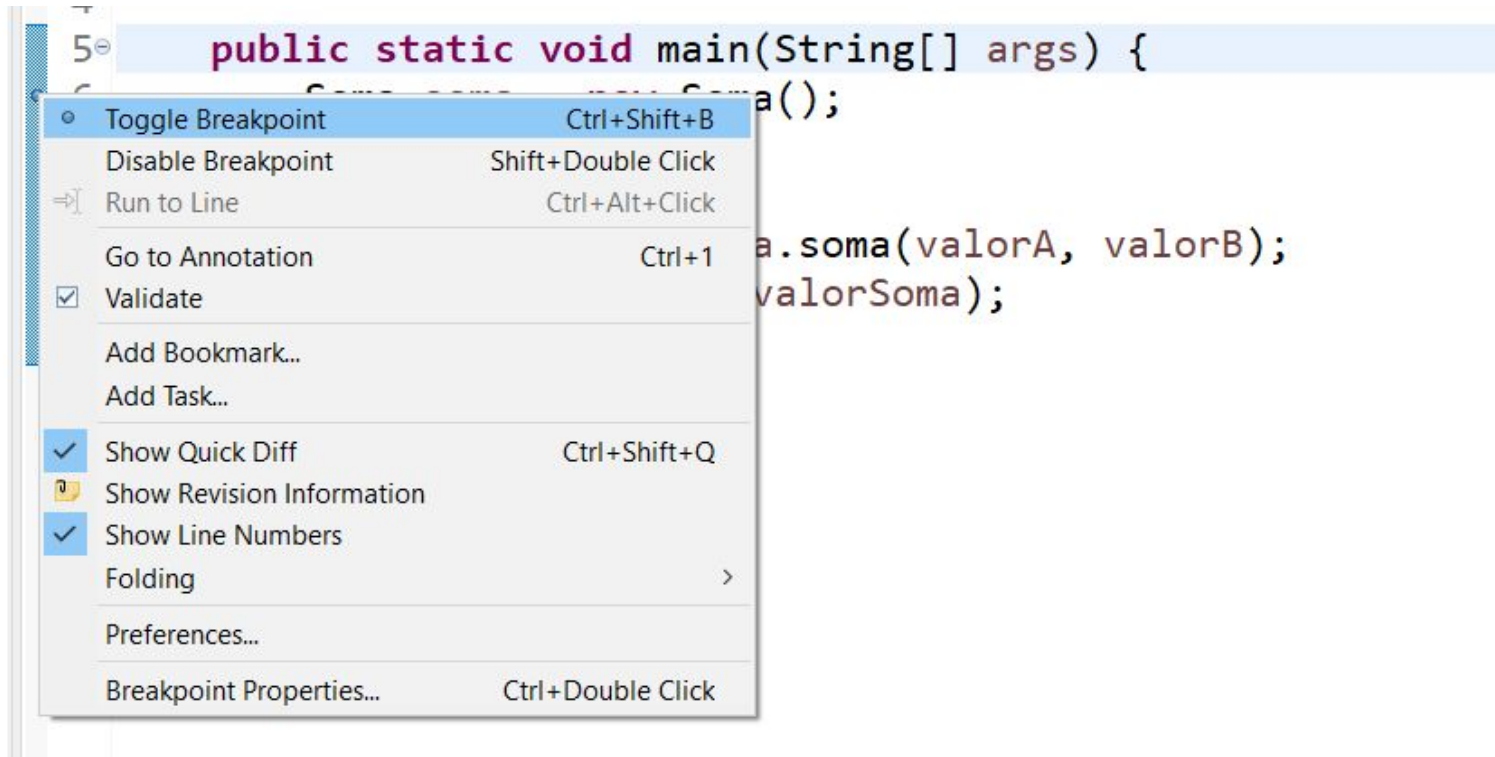
# Dicas para encontrar os defeitos

- Perspectiva de Debug no Eclipse



# Dicas para encontrar os defeitos

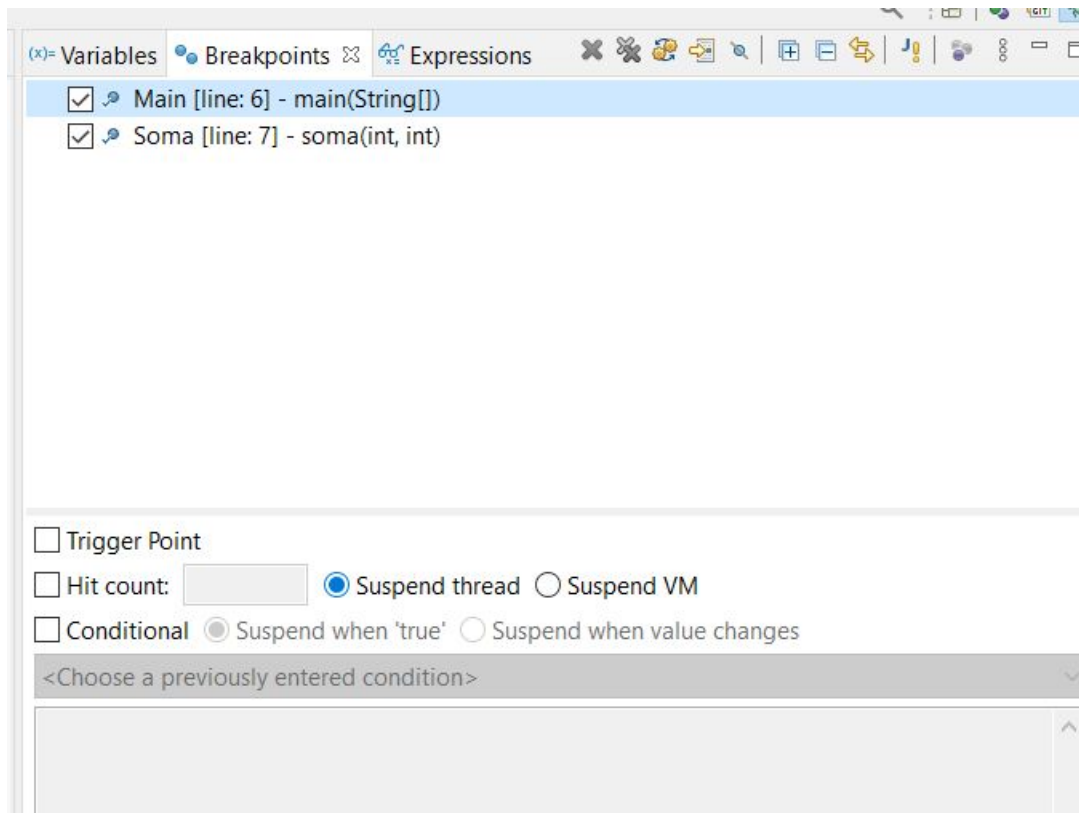
- Breakpoints
  - Marca o local do código onde a execução deve ser suspensa



# Dicas para encontrar os defeitos

- **Aba Breakpoints**

- Visualiza seus breakpoints e é possível desabilitá-los também



- **Hit Count**

- É possível restringir que o breakpoint só será ativado quando a linha em que ele encontra-se for executada 'X' vezes

- **Conditional**

- Pode definir uma condição para o breakpoint



## Dicas para encontrar os defeitos

- **Variáveis**

- **Exibe as variáveis da execução**

[illegible]

## Dicas para encontrar os defeitos

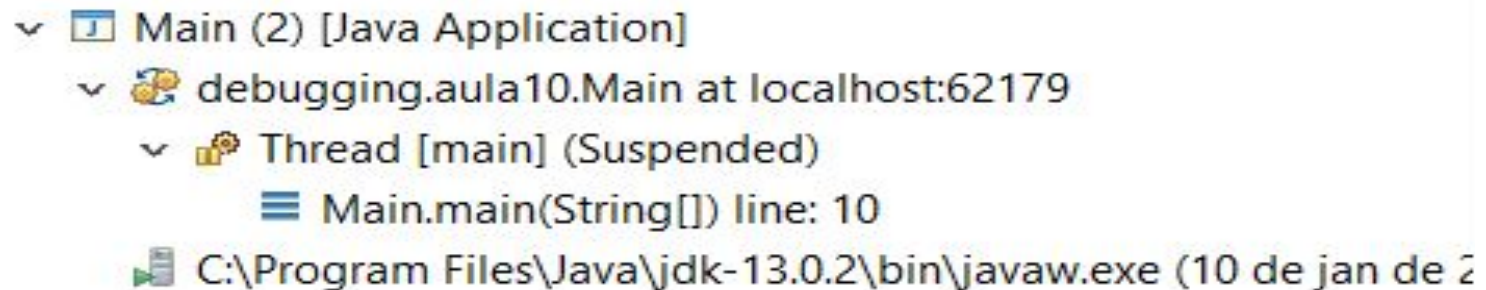
- **Expressions**

- **Possibilita verificar o valor de uma expressão**

[illegible]

# Dicas para encontrar os defeitos

- Threads
  - Threads que estão sendo executadas



The screenshot shows the 'Threads' window in a Java IDE. It displays a hierarchy of threads. At the top is 'Main (2) [Java Application]'. Under it is 'debugging.aula10.Main at localhost:62179'. Under that is 'Thread [main] (Suspended)'. Below the thread name, it shows the current execution point: 'Main.main(String[]) line: 10'. At the bottom, it shows the full path to the Java executable: 'C:\Program Files\Java\jdk-13.0.2\bin\javaw.exe (10 de jan de 2020)'.

```
▼ [J] Main (2) [Java Application]
  ▼ [🔗] debugging.aula10.Main at localhost:62179
    ▼ [🔧] Thread [main] (Suspended)
      [≡] Main.main(String[]) line: 10
      [📄] C:\Program Files\Java\jdk-13.0.2\bin\javaw.exe (10 de jan de 2020)
```

# Dicas para encontrar os defeitos

- Comandos
    - **F5** - Vai para o próximo passo do seu programa. Se for um método, ele entra nele
    - **F6** - Também vai para o próximo passo
    - **F7** - Voltará e mostrará o método que fez a chamada para o código que está sendo depurado
    - **F8** - Vai para o próximo breakpoint, se nenhum for encontrado, o programa seguirá seu fluxo de execução normal.
-

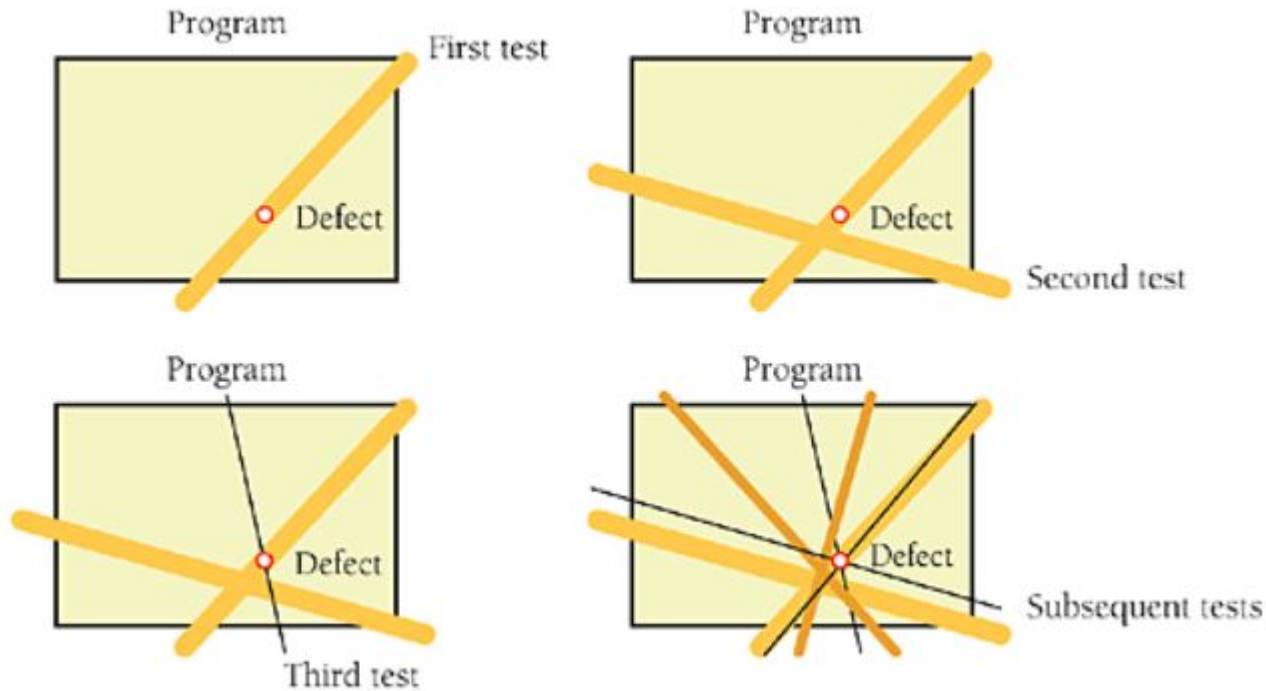
# Dicas para encontrar os defeitos

- As ferramentas de debugger permitem
  - Parar a execução quando
    - atingem uma linha específica
    - quando uma variável global muda
    - quando uma variável assume um determinado valor
  - Executar linha por linha
  - Verificar os valores das variáveis naquele momento



# Dicas para encontrar os defeitos

- Reproduza o erro de diferentes formas
  - Triangulação do defeito



# Dicas para encontrar os defeitos

- Gere mais dados para gerar mais hipóteses
  - Crie casos de testes diferentes
- Pense em diferentes hipóteses possíveis
  - Não se limite a uma primeira hipótese
- Tente refinar a região suspeita do código com defeito
  - Use o debugger para pular chamadas de métodos
  - Pode-se usar dividir-e-conquistar para tentar localizar a região do defeito



# Dicas para encontrar os defeitos

- Verifique código alterado recentemente
  - Você pode comparar com uma versão antiga do sistema para verificar se o defeito acontecia na versão antiga também
  - Use log de controle de versão para identificar qual código foi alterado
- Suspeite de classes e métodos que tiverem defeitos antes
  - Essas classes e métodos tem uma maior probabilidade de ainda terem defeitos





# Erros de Sintaxe

- Estão cada vez mais simples de identificar devido aos diagnósticos dos compiladores atuais
- Porém,
  - Não confie nos números de linha nas mensagens do compilador
    - Análise também antes e depois da linha indicada
    - Tente verificar porque o compilador indicou a linha errada
  - Se vierem várias mensagens de erro, corrija a primeira e compile novamente
  - Aplique a técnica Dividir e conquistar
    - Remova/Comente parte do código e compile novamente

# Corrigindo um defeito

- Temos que ter cuidado ao corrigir um defeito para de fato corrigir o defeito e não adicionar novos defeitos
- Para reduzir as chances de erro:
  - Entenda o problema antes de corrigir ele
  - Entenda o programa e não só o problema
  - Crie casos de testes que confirmem sua hipótese
    - Eles também serão usados para validar a correção do problema
  - Corrija o problema e não o sintoma
    - Cuidado com correções condicionadas (SE ...)

# Corrigindo um defeito

- Mais algumas dicas:
  - Faça uma mudança por vez
    - Mais fácil de rastrear as alterações e a correção do bug
  - Verifique sua correção
    - Execute os casos de testes novamente
  - Adicione casos de testes unitários que expõem o defeito
    - Assim você saberá se ele voltar a acontecer no futuro
  - Procure por defeitos similares
    - Defeitos tendem a ocorrer em grupos!

# Como auxiliar na depuração durante desenvolvimento?

- Boa formatação
- Comentários
- Bons nomes de variáveis e métodos
- Use nomes de variáveis com grande ‘distância psicológica’
  - É a facilidade com que dois itens podem ser diferenciados

1º variável	2º variável	Distância psicológica
shiftrn	shiftrm	Quase nenhuma
dcount	bcount	Baixa
product	sum	Grande

# Checklist - Depuração

- Você utiliza depuração como uma oportunidade para aprender mais sobre seu programa, erros, qualidade de código e abordagens de resolução de problemas?
- Você evita abordagem de tentativa e erro na depuração?
- Você assume que os defeitos são do seu código?
- Você utilizar métodos científicos para encontrar defeitos?
- Você utiliza diferentes abordagens para encontrar defeitos?
- Você verifica que a correção está correta?
- Você utiliza as mensagens do compilador, testes e ferramentas de depuração?



# Trabalho Prático - TP5

- TRABALHO PRÁTICO - TP5 (Aula Prática 13/01/2020)
  - Simular um erro no código do projeto final
  - Usar algum depurador (*debugger*) para monitorar a execução do sistema até o local do bug
  - Utilizar **pelo menos duas** funcionalidades do depurador
  - Resposta no Google Classroom
    - Deve ser individual
    - Indicar nome do depurador utilizado
    - Prints com imagens do debugger enquanto você o utiliza
      - e.g., exibindo os valores das variáveis

**PS: Aproveitem também para evoluir o código do projeto final**

---

# Projeto Final

- **APF-2**
    - Entrega: 10/02/2020
    - Foco: MVP com testes automatizados (unitários e de sistema)
    - Formato: Código e Vídeo
  - **APF-3**
    - Entrega: 31/03/2020
    - Foco: Entrega Final do Projeto aplicando Clean Code, Programação Defensiva e Boas práticas de projeto
    - Formato: Código
  - **Apresentações dos projetos finais**
    - Dias 05 e 07 de abril de 2020
-

# *Obrigado!*

## *Por hoje é só pessoal...*

# **Dúvidas?**



qpg4p5x



ismaylesantos@great.ufc.br



@IsmayleSantos

---