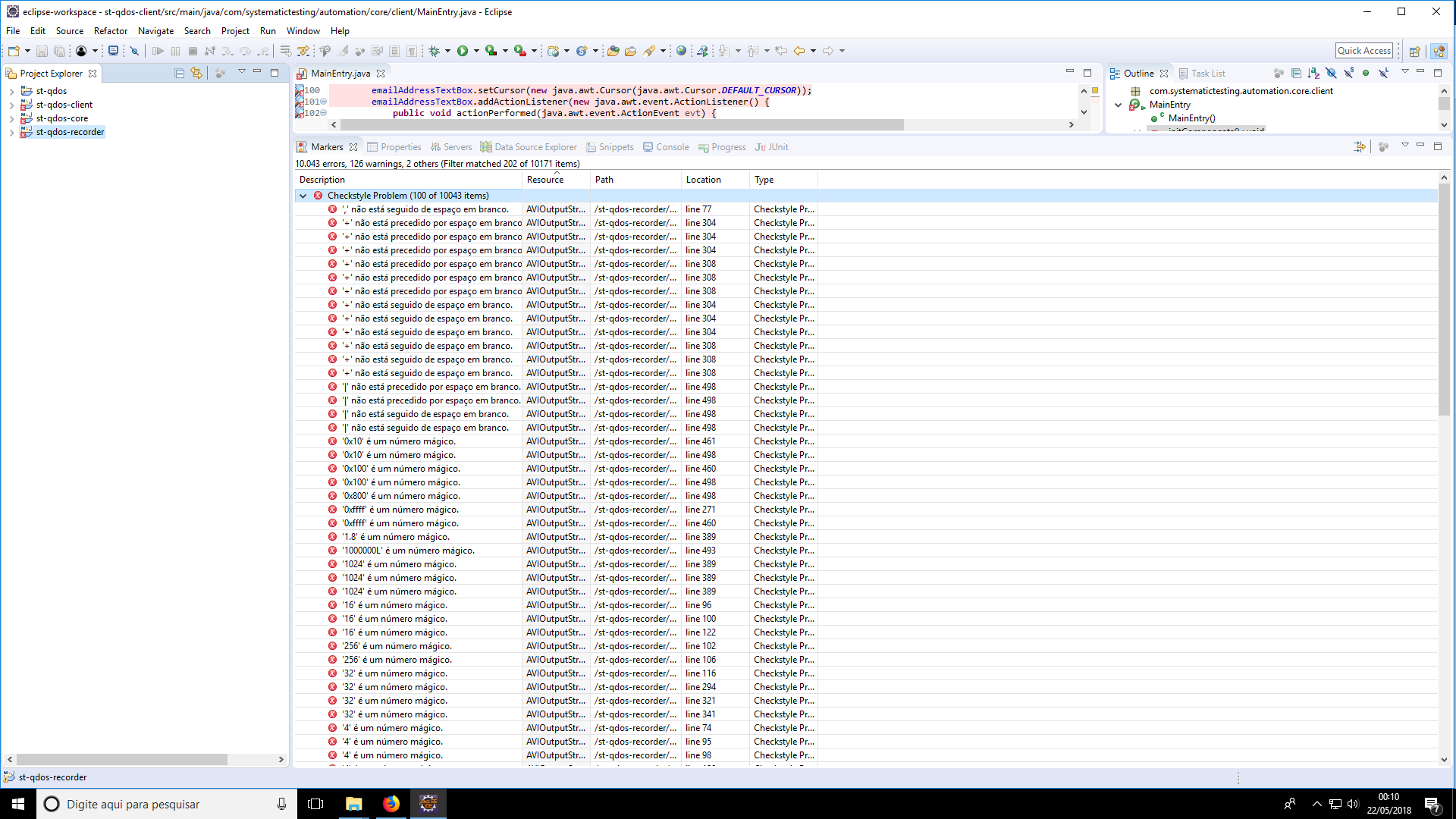
1. **Ferramenta CheckStyle**

O Checkstyle é uma ferramenta de desenvolvimento para ajudar programadores a escrever código Java que adere a um padrão de codificação. Ela automatiza o processo de verificação do código Java para poupar os humanos dessa tarefa chata (mas importante). Isso o torna ideal para projetos que desejam impor um padrão de codificação.

Checkstyle pode verificar muitos aspectos do código-fonte. Pode encontrar problemas de design de classe, problemas de design de método. Ele também tem a capacidade de verificar o layout do código e os problemas de formatação.

Executando o CheckStyle no projeto QDOS, obtemos 10043 erros, como mostra a figura abaixo:



A ferramenta encontrou as seguintes violações de estilo:

* **AbbreviationAsWordInName:** O Check validar abreviações (letras maiúsculas consecutivas) de comprimento no nome do identificador, também permite aplicar a nomenclatura de maiúsculas e minúsculas.
* **AbstractClassName:** Garante que os nomes das classes abstratas estejam em conformidade com alguma expressão regular e verifique se existe um modificador abstrato.
* **AnnotationOnSameLine:** A verificação verifica se as anotações estão localizadas na mesma linha com seus destinos.
* **AnonInnerLength:** Verifica por longas classes internas anônimas.
* **ArrayTrailingComma:** Verifica se a inicialização da matriz contém vírgula à direita opcional.
* **ArrayTypeStyle:** Verifica o estilo das definições de tipo de matriz.
* **AtclauseOrder:** Verifica a ordem das cláusulas.
* **AvoidEscapedUnicodeCharacters:** Restrinja usando escapes Unicode.
* **AvoidInlineConditionals:** Detecta condicionais inline.
* **AvoidNestedBlocks:** Localiza blocos aninhados.
* **AvoidStarImport:** Verifique se encontra instruções de importação que usam a notação \*.
* **ComentáriosIndentação:** Controla o recuo entre comentários e código circundante.
* **DescendantToken:** Verifica se há tokens restritos abaixo de outros tokens.
* **DesignForExtension:** Verificações projetadas para herança.
* **EmptyBlock:** Verifica blocos vazios, mas não válida blocos sequenciais.
* **EmptyCatchBlock:** Verifica se há blocos de captura vazios com poucas opções para ignorar violação.
* **EmptyForInitializerPad:** Verifica o preenchimento de um vazio para inicializador; isto é, se um espaço é necessário para um inicializador vazio ou se tais espaços são proibidos.
* **EmptyForIteratorPad:** Verifica o preenchimento de um vazio para o iterador; isto é, se um espaço é necessário em um espaço vazio para o iterador, ou se tais espaços são proibidos.
* **EmptyLineSeparator:** Verifica se há separadores de linha em branco.
* **EmptyStatement:** Detecta instruções vazias (independentes ';').
* **FileLength:** Verifica arquivos de origem longos.
* **FileTabCharacter:** Verifica se um arquivo contém um caractere de tabulação.
* **FinalClass:** Verifica se a classe somente construtores privados é declarada como final.
* **FinalLocalVariable:** Garante que as variáveis ​​locais que nunca tiveram seus valores alterados devem ser declaradas como finais.
* **FinalParameters:** Verifique se os parâmetros method / constructor / catch / foreach são finais.
* **HiddenField:** Verifica se uma variável local ou um parâmetro não ofusca um campo definido na mesma classe.
* **HideUtilityClassConstructor:** Certifique-se de que as classes de utilitário (classes que contêm apenas métodos estáticos) não tenham um construtor público.
* **InterfaceIsType:** Implementa Bloch, Effective Java, Item 17 - Use Interfaces somente para definir tipos.
* **InterfaceTypeParameterName:** Verifica se os nomes de parâmetro do tipo de interface estão em conformidade com um formato especificado pela propriedade format.
* **JavadocMethod:** Verifica o Javadoc de um método ou construtor.
* **JavadocStyle:** Custom Checkstyle Marque para validar o Javadoc.
* **JavadocTagContinuationIndentation:** Verifica o recuo das linhas de continuação em cláusulas.
* **JavadocType:** Verifica o Javadoc de um tipo.
* **JavadocVariable:** Verifica se uma variável tem comentário de Javadoc.
* **JavaNCSS:** Esta verificação calcula a métrica NCSS (Non Commenting Source Statements) para arquivos e métodos de origem Java.
* **LeftCurly:** Verifica o posicionamento das chaves no tipos, métodos e outros blocos:
* **LineLength:** Verifica por linhas longas.
* **LocalFinalVariableName:** Verifica se os nomes finais da variável local estão em conformidade com um formato especificado pela propriedade format.
* **LocalVariableName:** Verifica se os nomes de variáveis ​​locais e não finais estão em conformidade com um formato especificado pela propriedade format.
* **MagicNumber:** Verifica números mágicos.
* **MemberName:** Verifica se os nomes das variáveis ​​de instância estão em conformidade com um formato especificado pela propriedade format.
* **MethodCount:** Verifica o número de métodos declarados em cada declaração de tipo por modificador de acesso ou contagem total.
* **MethodLength:** Verifica por métodos longos.
* **MethodName:** Verifica se os nomes dos métodos estão em conformidade com um formato especificado pela propriedade format.
* **MissingCtor:** Verifica que as classes (exceto a abstrata) definem ctor e não dependem do padrão.
* **MissingDeprecated:** Esta classe é usada para verificar se a anotação java.lang.Deprecated está presente e se a tag Javadoc @deprecated está presente quando ela está presente.
* **MutableException:** Garante que as exceções (definidas como qualquer nome de classe em conformidade com alguma expressão regular) sejam imutáveis.
* **NeedBraces:** Verifica se há chaves em volta dos blocos de códigos.
* **NewlineAtEndOfFile:** Verifica se há uma nova linha no final de cada arquivo.
* **NoClone:** Verifica se o método clone não é substituído da classe Object.
* **NoFinalizer:** Verifica que nenhum método com zero parâmetros é definido usando o nome finalizado.
* **NoLineWrap:** Verifica se as instruções escolhidas não são preenchidas com linhas.
* **NonEmptyAtclauseDescription:** Verifica se a tag de cláusula é seguida pela descrição.
* **NPathComplexity:** Verifica a complexidade do npath em relação a um limite especificado (padrão = 200).
* **OneTopLevelClass:** Verifica se cada classe de nível superior, interfaces ou enum reside em um arquivo de origem próprio.
* **OperatorWrap:** Verifica a quebra de linha para os operadores.

1. **Ferramenta Error Prone**

O Error Prone é uma ferramenta de análise estática para Java que detecta erros comuns de programação em tempo de compilação. Usando o Error Prone para aumentar a análise de tipo do compilador, pode-se pegar mais erros antes que eles lhe custem tempo ou acabem como erros na produção.

Executando o Error Prone no projeto QDOS, não foi reportado nenhum erro pela ferramenta, como mostra a figura abaixo:

