Web-based Center for Automated Testing
- Web automated grader for programming assignments.
- Plug-in-based architecture to support TDD.

Components
- Checkstyle
  - Code documentation/layout standard checking tool.
- PMD (Programming Mistake Detector)
  - Code analysis tool to detect potential problems.
- TDD Grader Subsystem.
  - Automates execution of students tests and instructor reference tests
Checkstyle

- Coding standard
  - Automates checking of code documentation and layout.

- Code checks ([http://checkstyle.sourceforge.net/checks.html](http://checkstyle.sourceforge.net/checks.html))
  - Checks Javadoc comments
  - Naming conventions for identifiers.
  - Source file header checks
  - Size: file length, line length, method length, number of parameters
  - Whitespace padding
  - Modifier order (public, private, static, final, etc.)
  - Block checks
    - Checks for empty blocks (empty if or else blocks)
    - Requires braces: all control statements must use `{   }`
    - Checks for unnecessary nested blocks

- BlueJ Integration
  - In BlueJ IDE: Tools menu, Checkstyle option – executes Checkstyle locally
Checkstyle (continued)

- General code checks
  - Inline conditionals: ternary operator \((\text{cond}) \ ? \ \text{expr1} \ : \ \text{expr2}\)
  - Covariant equals() : must over-ride \text{equals}(\text{Object})
  - Empty statements ; ;
  - Hash code : if equals is over-ridden must over-ride \text{hashCode}()
  - Final local variables – variables whose values are never changed
  - Hidden field – local variables/parameters with same name as instance variables
  - Inner assignments – assignments nested in subexpressions \(x = (y = x+y)\);
  - Magic numbers – literals used that are not defined constants
  - Switch statement default clause exists
  - Switch statement default clause is last
  - Switch fall through – checks for cases clauses that lacks a fall through
  - Loop control variable modified – for loop variable changed in loop

**WARNING:** every code check performed by checkstyle is **NOT** listed in these notes.
Checkstyle (continued)

- General code checks
  - Complicated boolean expressions
  - Complicated boolean return statements
  - String literal equality: checks for condition of the form `if (name == "bob")`
  - Nested if depth limit
  - Return statement number limit in methods
  - Unused class type checks
  - Declaration Order: static class variables, instance variables, Constructors, methods
  - Parameter assignment check
  - Explicit Initialization – checks if default values used to initialize instance variables
  - Constructor existence check – requires classes to define at least 1 constructor
  - Multiple string literal checks
  - Multiple variable declarations – requires 1 declaration per line
  - Unnecessary parenthesis
PMD

- Code analysis tool

- General code problems checked:
  - Empty statement clauses
  - Unused code: local variables and parameters not accessed, private methods not invoked.
  - Unnecessary if statements
  - For loops that should be while loops & vice versa
  - Duplicated code

- Specific code problems checked:
  - Empty constructor, but at least 1 constructor
  - Null assignment after initialization
  - Multiple return statements
  - Assignments in actual operands
  - Base class constructor not called super()
  - SingularField: A field that's only used by one method could perhaps be replaced by a local variable

**WARNING:** every code check performed by PMD is **NOT** listed in these notes. There is overlap between the PMD & checkstyle checks.
Specific code problems checked:
- Boolean Inversion: `boolean b; ... b = !b;` should be coded `b ^= true;`
- Large number of imports (indicates possible high coupling)
- Local Variable Could Be Final: A local variable assigned only once can be declared final.
- Instantiating Objects In Loops
- Empty control statements: if, while, for, switch
- Unnecessary Return statements
- Unconditional If Statement: "if" statements that are always true or always false.
- EqualsNull: do not use equals() to compare to null.
- SimplifyConditional: No need to check for null before an instanceof; the instanceof keyword returns false when given a null argument.
- CompareObjectsWithEquals: Use equals() to compare object references; avoid comparing them with `==`.
- StringToString: Avoid calling `toString()` on String objects; unnecessary
- High Cyclomatic Complexity – number of unique execution paths through a method should be small
Specific code problems checked:

- Excessive Public Count: large number of public methods in a class may indicate class needs to be broken up
- Too Many Fields: Classes that have too many instance fields could be redesigned to have fewer fields,
- Duplicate Imports: Avoid duplicate import statements.
- Unused Imports: Avoid unused import statements.
- Don’t Import Java Lang: Avoid importing from package 'java.lang‘, these classes are automatically imported
- Short variable names & very long variable names
- Method Naming Conventions: Method names should always begin with a lower case character, and should not contain underscores.
- Class Naming Conventions: Class names should always begin with an upper case character.
- JUnit Tests Should Include Assert: JUnit tests should include at least one assertion.
Web-CAT Testing

- Automated code checking, analysis, and TDD evaluation system
  - http://web-cat.cs.vt.edu

- Web-CAT Grader TDD Subsystem
  - Compiles submitted student code and student test classes.
  - Executes submitted tests & code to ensure tests pass.
  - Performs code coverage of the test case code to determine which parts of a software system are not being tested.
  - Measures statement coverage, branch coverage & method coverage.
    - Clover: code coverage plug-in used by Web-CAT
  - Compiles students system code with instructor reference test case code.
  - Executes student code and instructor test code to determine reference tests passed.
class Person

Problem

- Submit the Person class and tests to Web-CAT.