Design by contract (DBC) is a software correctness methodology. It uses preconditions and postconditions to document the change in the state caused by a piece of a program. The main idea behind is that a class and its client have a contract with each other. This specifies preconditions a client must guarantee when calling a method defined by its class and postconditions in return that class guarantees after the call is held. A violation occurs while a program is running can be detected immediately because these contracts are defined in program code into the programming language itself and are translated into executable code by the compiler.

DBC is better for documentation than using program comments or Javadoc comments because Java modeling language (JML) strengthen such process without requiring formalization on them. JML allows informal descriptions within specification and considers them as a Boolean expression. Such permitting couples informal descriptions with formal statements, so that makes these statements more clear. The other reason is that a formal specification in JML is machine-checkable, so that can keep them up to date and help with debugging errors in the code.

spec_public declares a protected or private variable public for specification purposes. JML needs a "specific public" attribute to enforces information hiding by ensuring that public specifications can only mention publicly visible names, which means that private fields are not allowed to be used in public specifications. Thus, if Java considers some fields to be private, then by using the annotation "spec public" in the declaration of the fields they would be only treated as publicly visible in JML. I would say a better way to handle this issue is to alter JML rules and allow it to use private fields in public specifications. In such a way, the constructor, methods, and invariants would have private and public visibility. They could specify private visible names and public visible names.

The JML\invariant expressions allowed to be violated in helper methods and constructors, where the keyword helper issued as a modifier to free methods and constructors from the obligation of stratifying invariants. It also can be violated when the control is inside the object's methods. The paper mentioned such helper methods must be private and can be used to avoid duplication of code, even code that does not start or end in a state that satisfies a type's invariants. Therefore the violation of invariant expressions makes sense here because avoiding duplication of code is the purpose of using the helper method. Also, it makes sense at some point in a program to violate an invariant, especially if such invariant is already broken for a temporary period inside the method.