In **Java 8 (and all versions of Java), Generics cannot be used in the throws or throw clause**. This is because Java uses **Type Erasure**, meaning that generic type information is removed at runtime. Since exceptions are part of Java's runtime mechanism, allowing generics in throw or throws could lead to **runtime type ambiguity**.

**Why Generics Cannot Be Used with throws?**

1. **Java does not allow generic types as exceptions.**
   * This is because at runtime, Java needs to know the exact type of an exception to handle it properly.
2. **Type erasure removes generic type information at runtime.**
   * Java cannot determine the exact type of a generic exception during exception handling.
3. **Exception handling relies on exact types, not parameterized types.**
   * The catch block works with known exception types, and generics introduce uncertainty.

**❌ Invalid Example: Using Generics in throws Clause**

// ❌ Compilation Error: Cannot throw or declare a generic exception

class GenericException<T> extends Exception {

private T data;

public GenericException(T data) {

this.data = data;

}

public T getData() {

return data;

}

}

class Test {

// ❌ Compilation Error: Cannot declare throws GenericException<T>

public <T> void riskyMethod() throws GenericException<T> {

throw new GenericException<>("This will not compile");

}

}

**🔴 Why does this fail?**

* The compiler **does not know what T is** at runtime because of **Type Erasure**.
* The JVM cannot differentiate between GenericException<String> and GenericException<Integer> when catching them.

**✔ Correct Approach: Use a Non-Generic Exception**

Instead of making the exception generic, store the generic data **inside** a non-generic exception.

// ✅ Correct: Define a normal exception and store generic data inside it

class CustomException extends Exception {

private Object data; // Use Object to hold generic data

public CustomException(Object data) {

this.data = data;

}

public Object getData() {

return data;

}

}

class Test {

public <T> void riskyMethod(T value) throws CustomException {

throw new CustomException(value); // Store generic data inside

}

public static void main(String[] args) {

Test test = new Test();

try {

test.riskyMethod("Java 8 Generics"); // Throws exception

} catch (CustomException e) {

System.out.println("Caught Exception: " + e.getData());

}

}

}

**✅ Why does this work?**

* The exception class (CustomException) itself is **not generic**.
* We store generic data inside Object, which allows flexibility.

**Conclusion**

1. **You cannot use generics in throws or throw clauses.**
2. **A workaround is to store generic data inside a non-generic exception class.**
3. **Type erasure prevents the JVM from knowing the generic type of exceptions at runtime.**