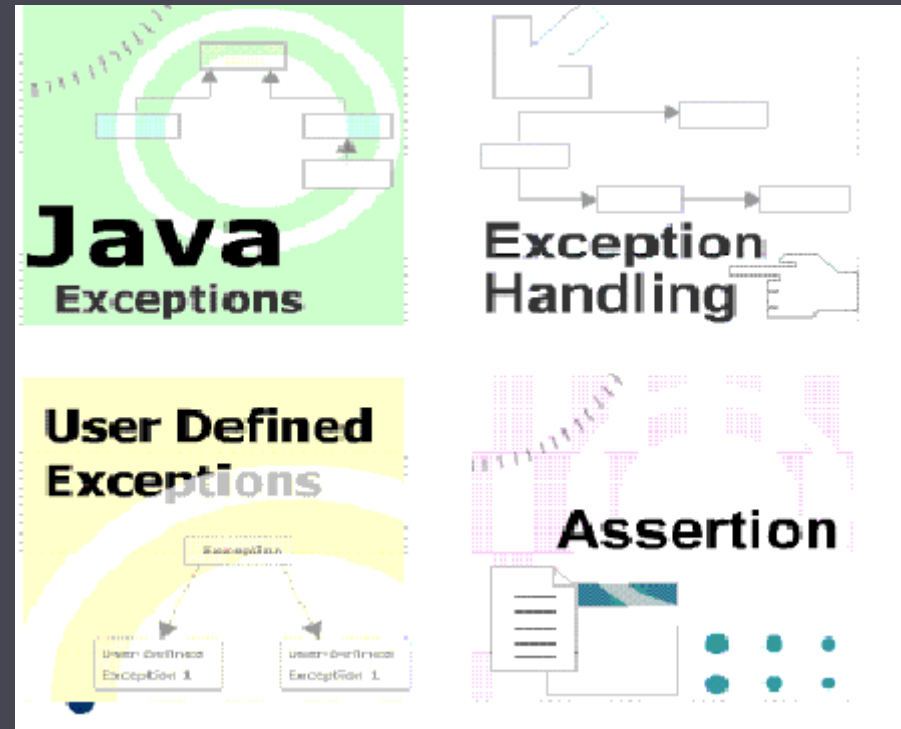


Module 9 & Overview

Exceptions & Overview

Module 9 - Exceptions

- Introduction to Exceptions
- Exception handling in Java
- User defined exceptions
- Assertions

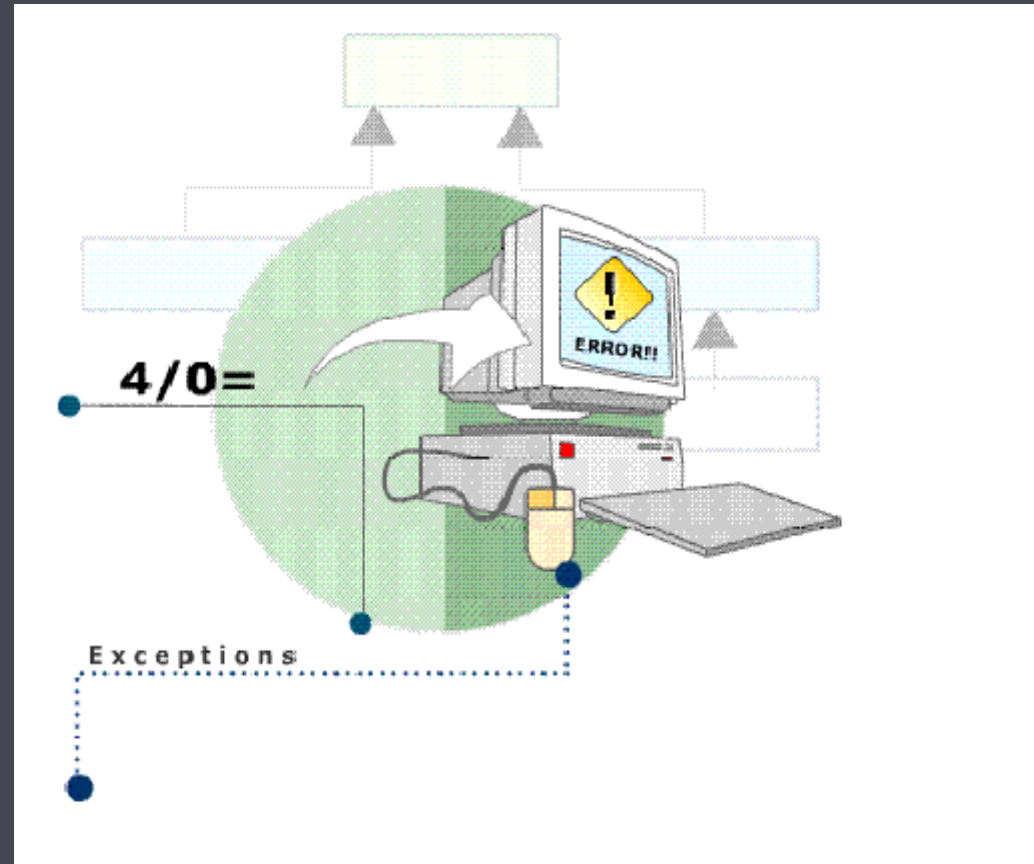


Introduction to Exceptions

- Explain the concept of Exceptions
- Identify the different types of Exceptions
- Knowledge Check

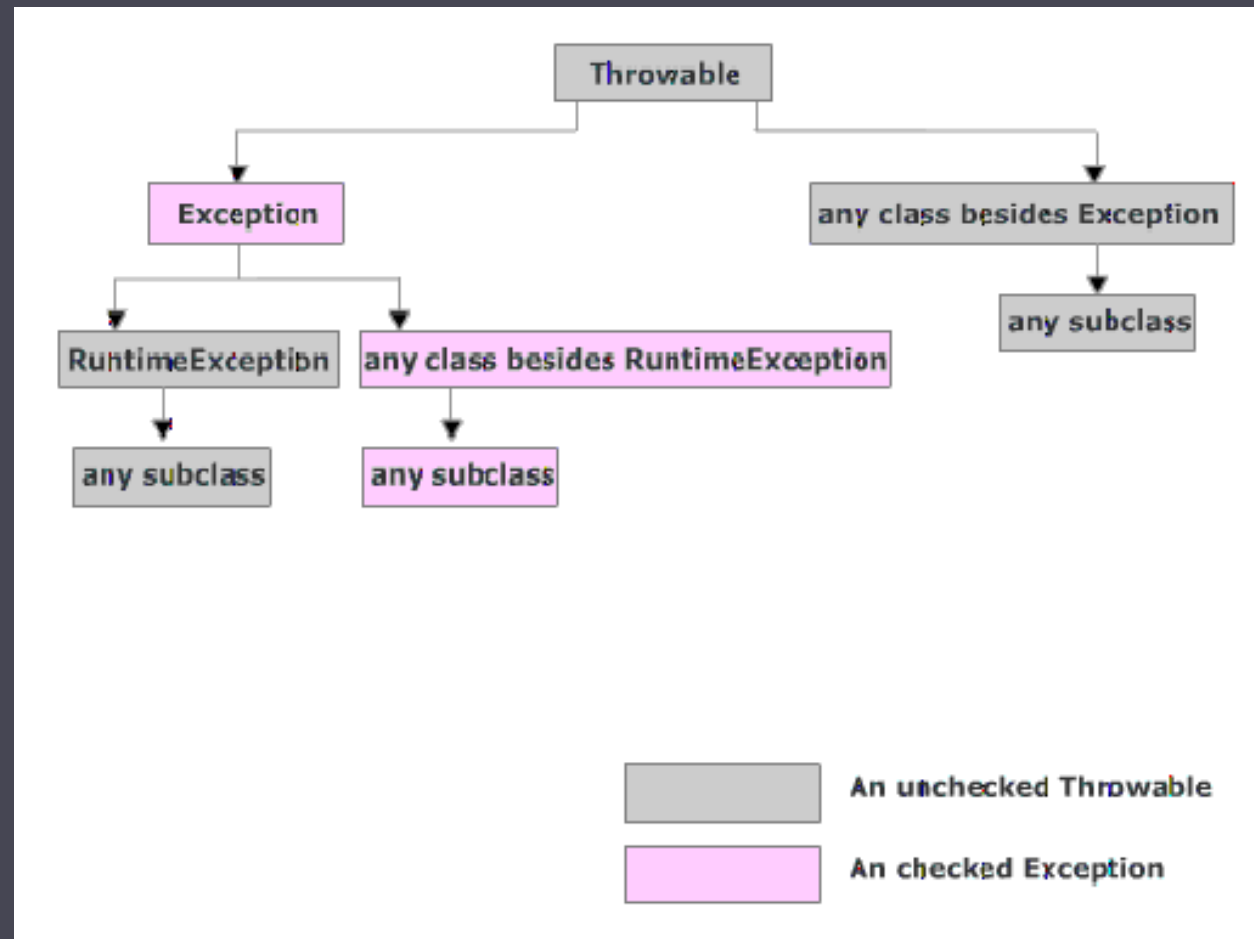
Causes for Exceptions

- Program errors
- Client code errors
- Errors beyond the control of a program



Classification of Exceptions

- Checked Exceptions
- Unchecked Exceptions



Types of Checked Exceptions

Exception	Description
InstantiationException	This exception occurs if an attempt is made to create an instance of the abstract class.
InterruptedException	This exception occurs if a thread is interrupted.
NoSuchMethodException	This exception occurs if the Java Virtual Machine is unable to resolve which method is to be called.
RuntimeException	This exception may be thrown during normal operation of Java Virtual Machine if some erroneous condition occurs.

Types of Unchecked Exceptions

Exception	Description
ArithmeticException	Derived from <code>RuntimeException</code> and indicates an Arithmetic error condition.
ArrayIndexOutOfBoundsException	Derived from <code>IndexOutOfBoundsException</code> class and is generated when an array index is less than zero or greater than the actual size of the array.
IllegalArgumentException	Derived from <code>RuntimeException</code> . Method receives an illegal argument.
NegativeArraySizeException	Derived from <code>RuntimeException</code> . Array size is less than zero.
NullPointerException	Derived from <code>RuntimeException</code> . Attempt to access a null object member.
NumberFormatException	Derived from <code>IllegalArgumentException</code> . Unable to convert the string to a number.
StringIndexOutOfBoundsException	Derived from <code>IndexOutOfBoundsException</code> . Index is negative or greater than the size of the string.

Exception handling in Java

- State the use of *try-catch* block
- Describe the use of *finally* block
- State the flow of execution in an exception handling block
- Describe the use of *throw* and *throws* keyword
- Describe the use of multiple *catch* block

The use of *try-catch* block: **Syntax**

```
try {  
    Statement_1;  
    Statement_2;  
    ...  
}  
catch(ExceptionType ObjectName) {  
    Statement_1;  
}
```

The use of *finally* block: **Syntax**

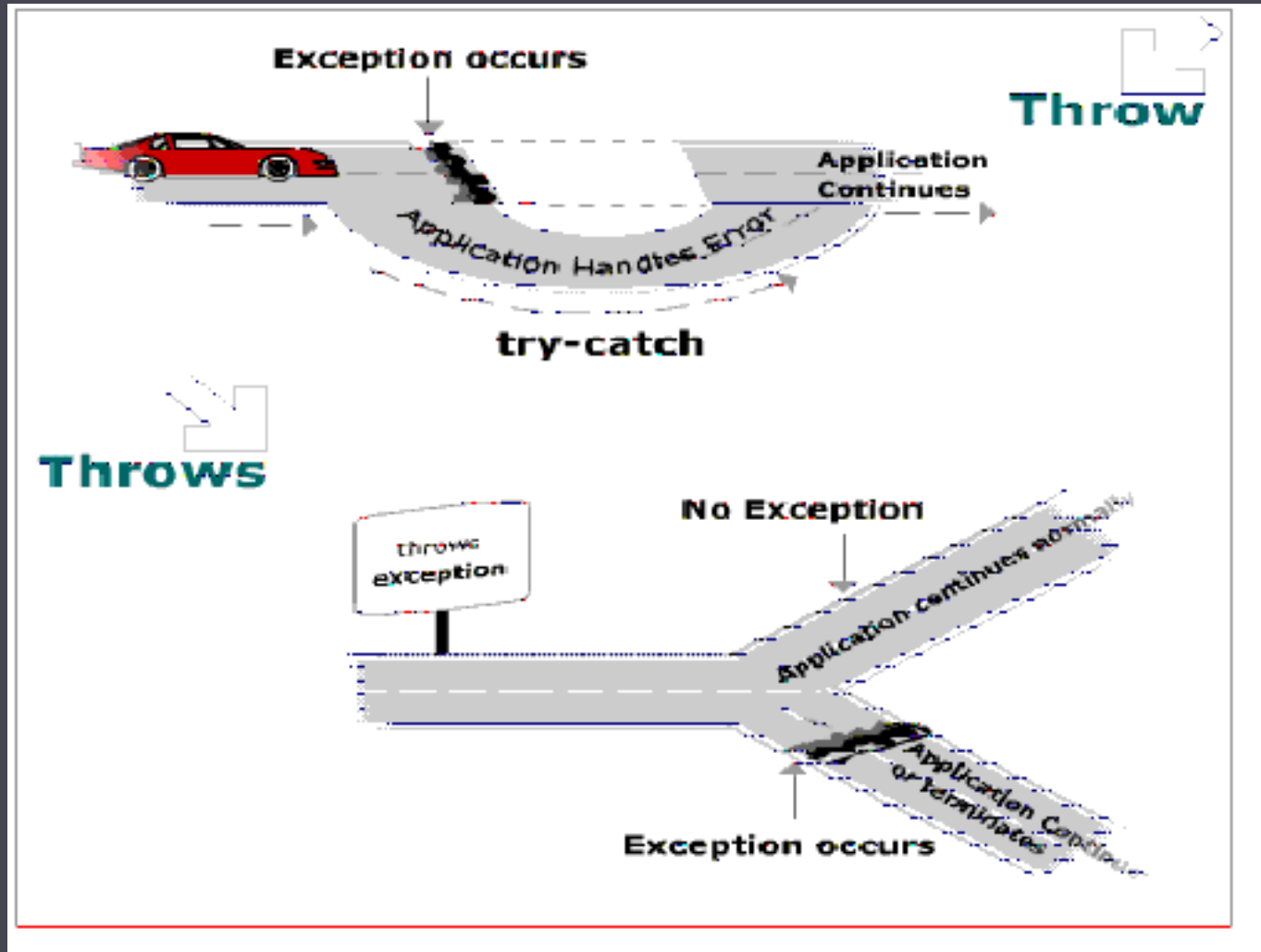
```
try {  
    Statement_1;  
    Statement_2  
}  
catch(ExceptionType ObjectName) {  
    Statement_1;  
}  
finally{  
    //Clean up code  
    Statement_1;  
}
```

The flow of execution in an exception handling block

- Execution Flow in case of Exception
 - ✓ Exception Object
 - ✓ Exception thrown
 - ✓ Stack trace
- Execution Flow with *try-catch* blocks
 - ✓ try-catch block
 - ✓ Exception caught
 - ✓ Call stack

The use of *throw* and *throws* keyword

- Syntax of *throw*: `throw ThrowableObject;`



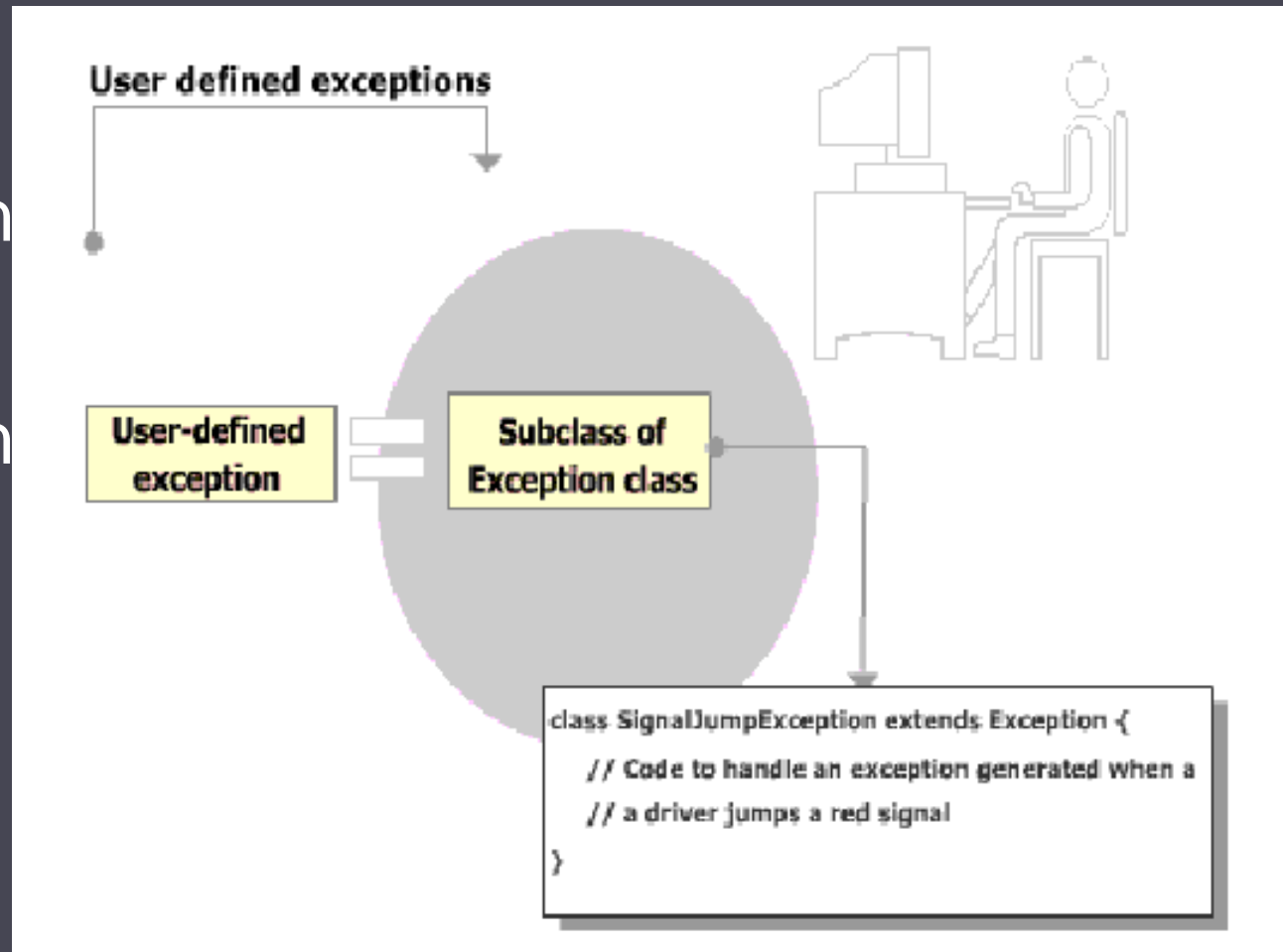
The use of multiple *catch* block

- Multiple *catch* block.
- Multiple *Exception*.

```
try{  
  
}  
catch (ExceptionType name){  
}  
catch (ExceptionType name){  
}
```

User defined exception

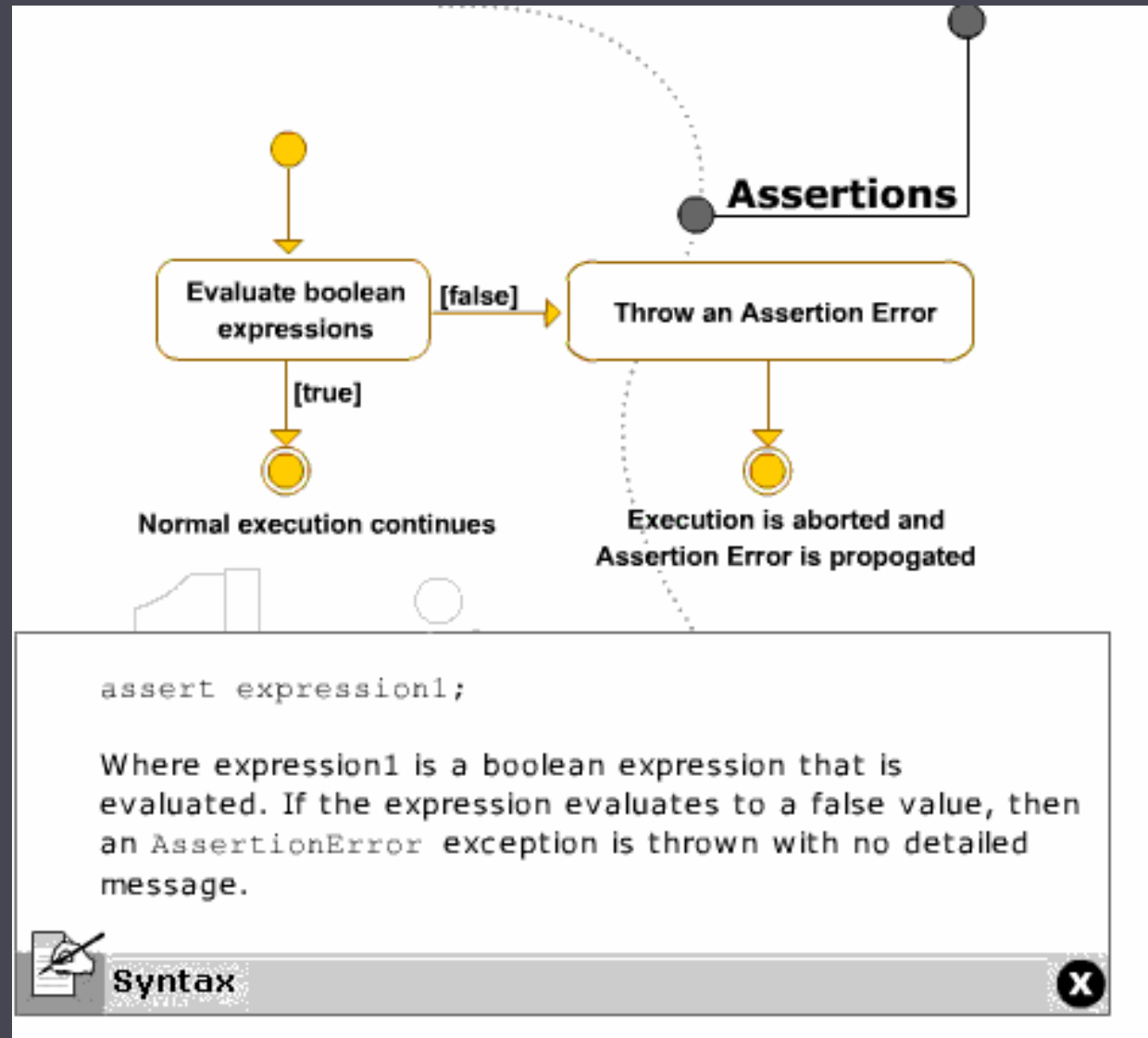
- Creating User Defined Exception
- Throwing User Defined Exception



Assertions

- Using the *assert* statement
- Using Assertions to manipulate variables
- Enabling and Disabling Assertions
- Where and Why to use Assertions
- Inappropriate uses of Assertions
- Using internal and control-flow invariants
- Knowledge Check

Using the *assert* statement



Enabling and Disabling Assertions

✓ Enabling assertion

✓ `java -ea`

✓ `java -enableassertions`

✓ Disabling assertion

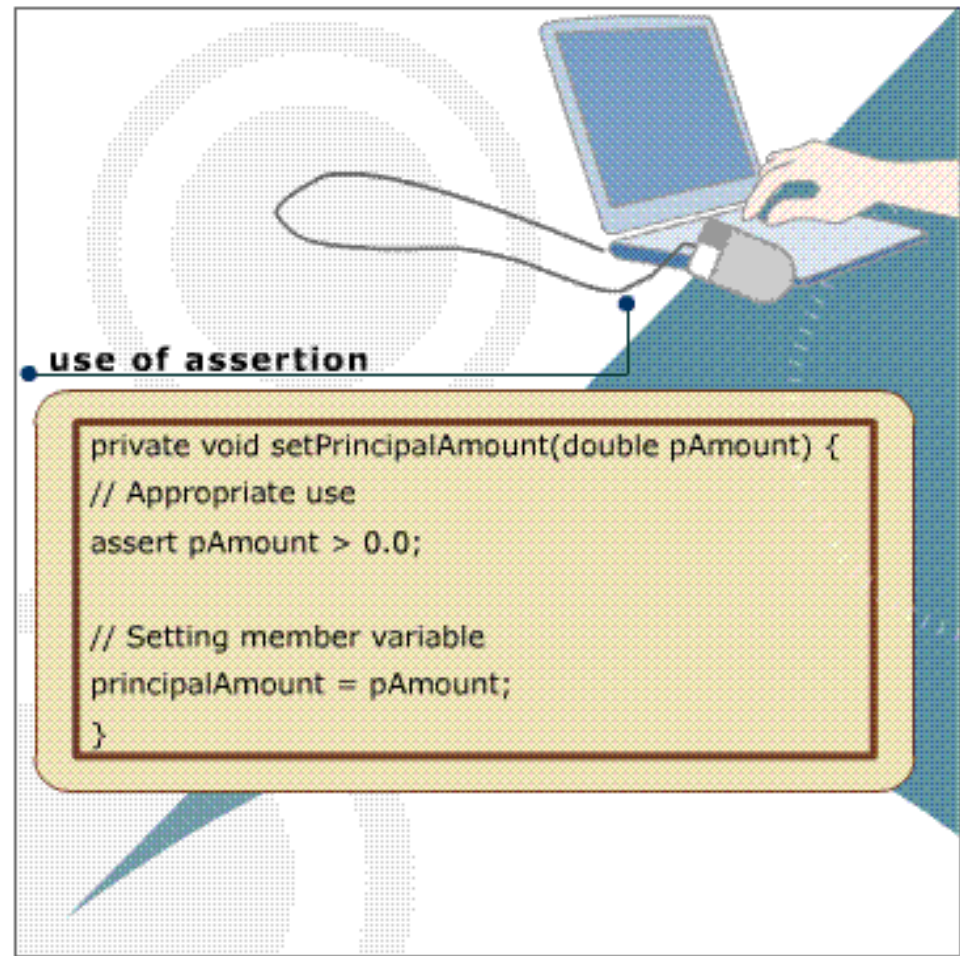
✓ `java -da`

✓ `java -disableassertions`

Where to use Assertions

Appropriate uses of assertions are:

- Arguments to private methods
- Conditions at the beginning of any method
- Conditional cases



Why to use Assertions

Assertion statements can be used:

- 👉 To replace comments.
- 👉 To replace code in default case.
- 👉 To replace unreachable code.
- 👉 In the beginning of the method.
- 👉 After method invocation.
- 👉 To check object's state.

Using Assertion to Test Unreachable Code

```
for (i=0;i<5;i++){  
    if (i==3)  
        return;  
    assert false:i;  
}
```

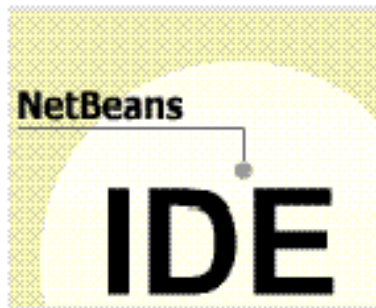
Workshop Review

- Use try-catch to handle exceptions
- Create and use user-defined exception class
- Use assertions to perform debugging in a Java program

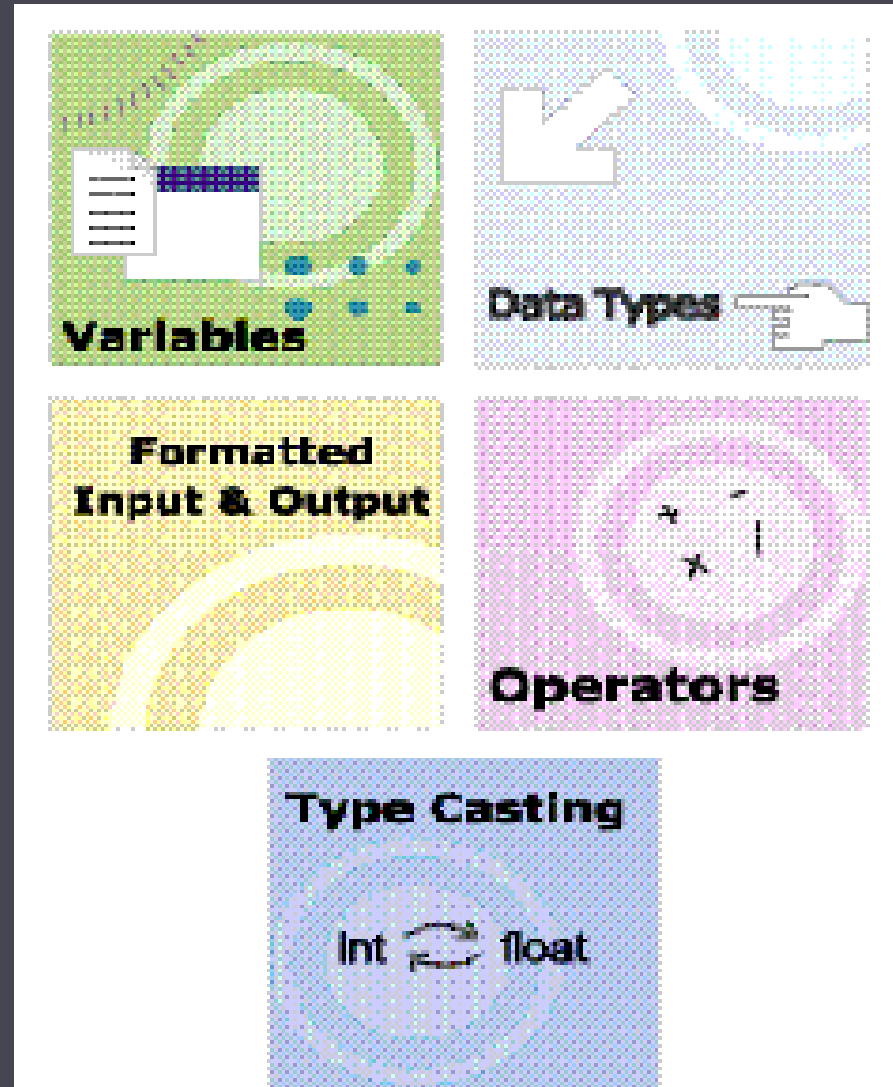
Final Module - Overview

- Introduction to Java
- Variables and Operators
- Decision-Making and Iterations
- Introducing Classes
- Arrays
- Packages and Access Specifiers
- Inheritance and Interface
- More on Classes
- Exceptions

Module 1- Introduction to Java



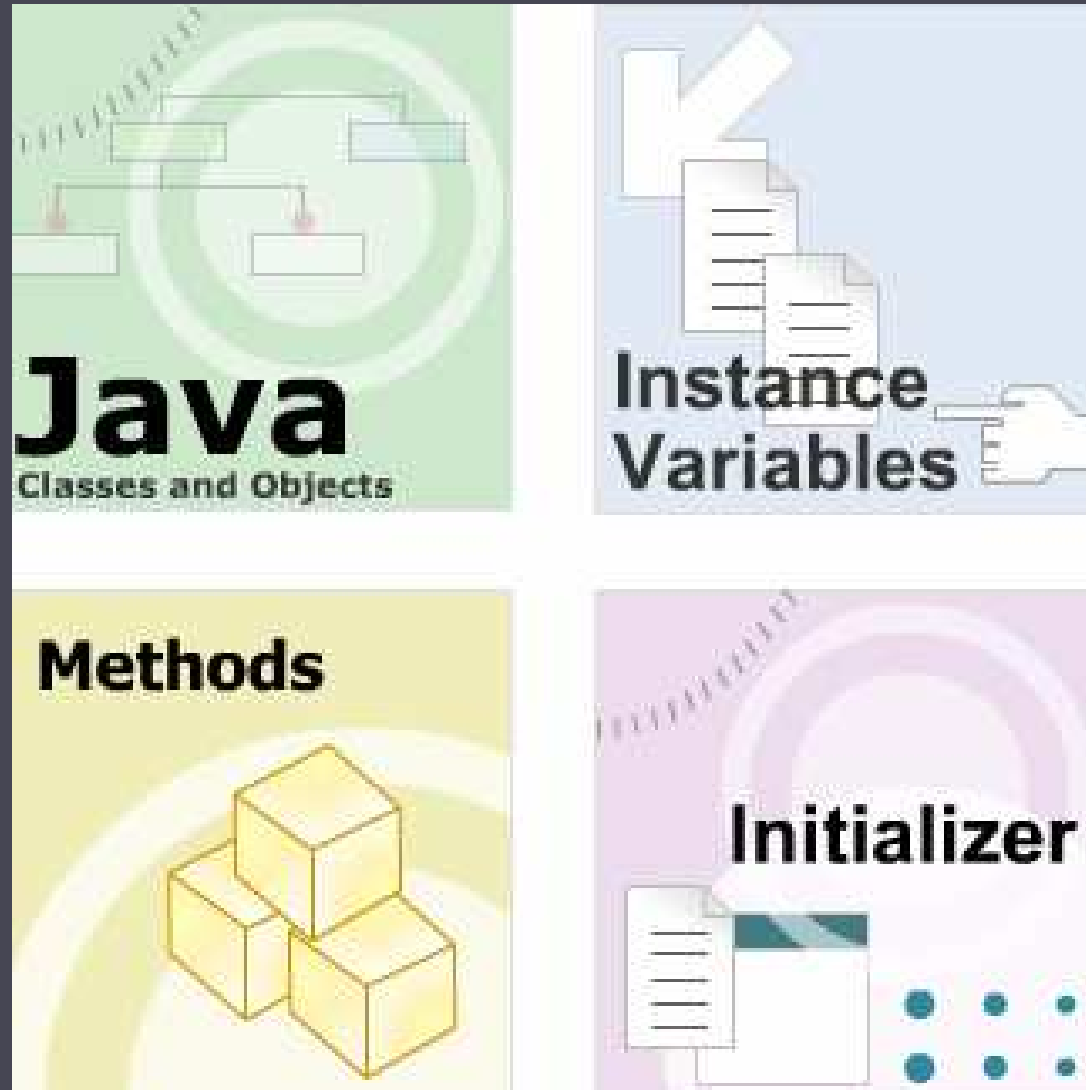
Module 2 - Variables and Operators



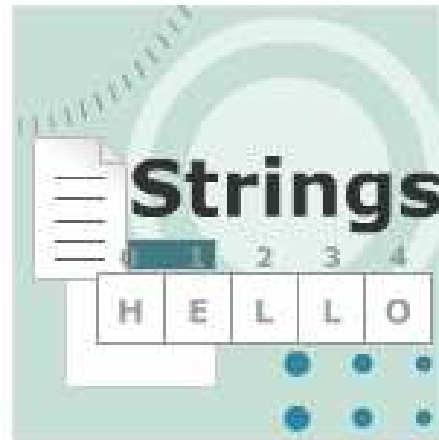
Module 3 - Decision-Making & Iterations



Module 4 - Introducing Classes



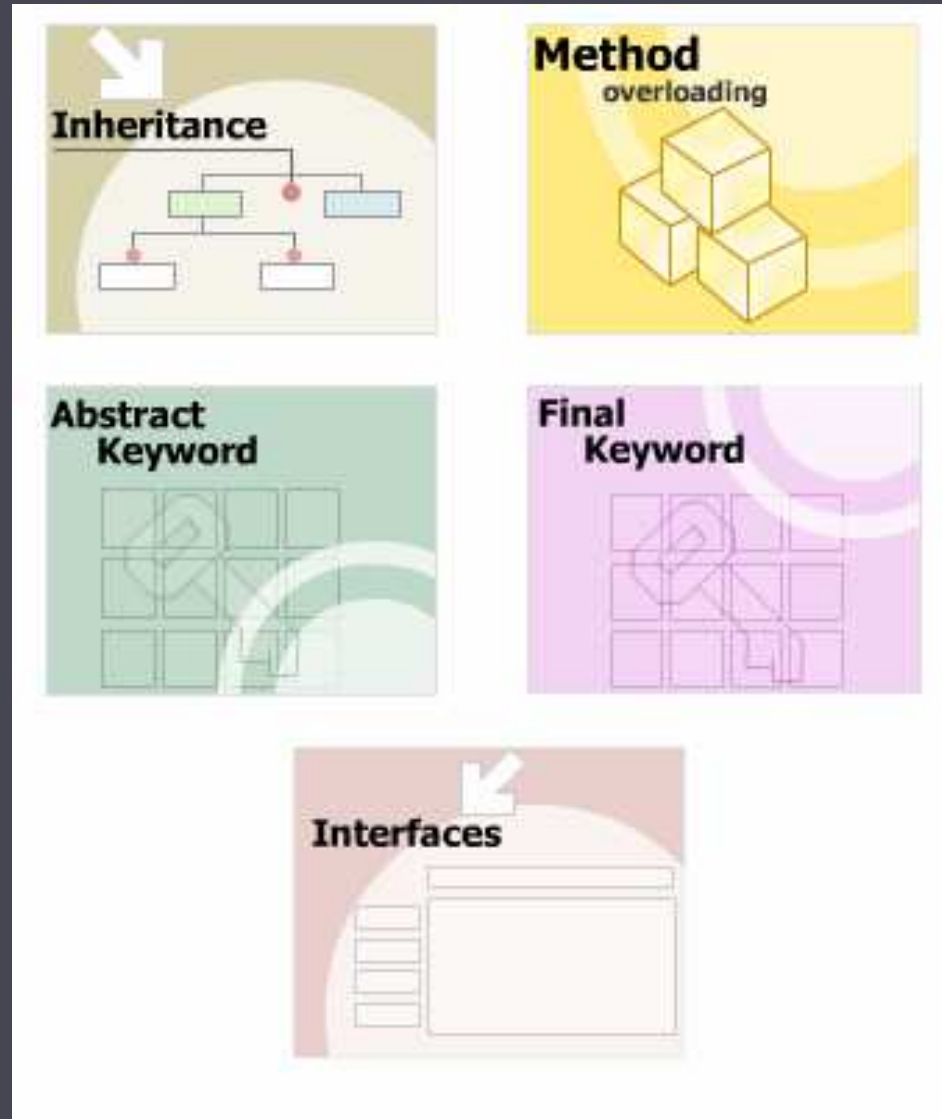
Module 5 - Arrays



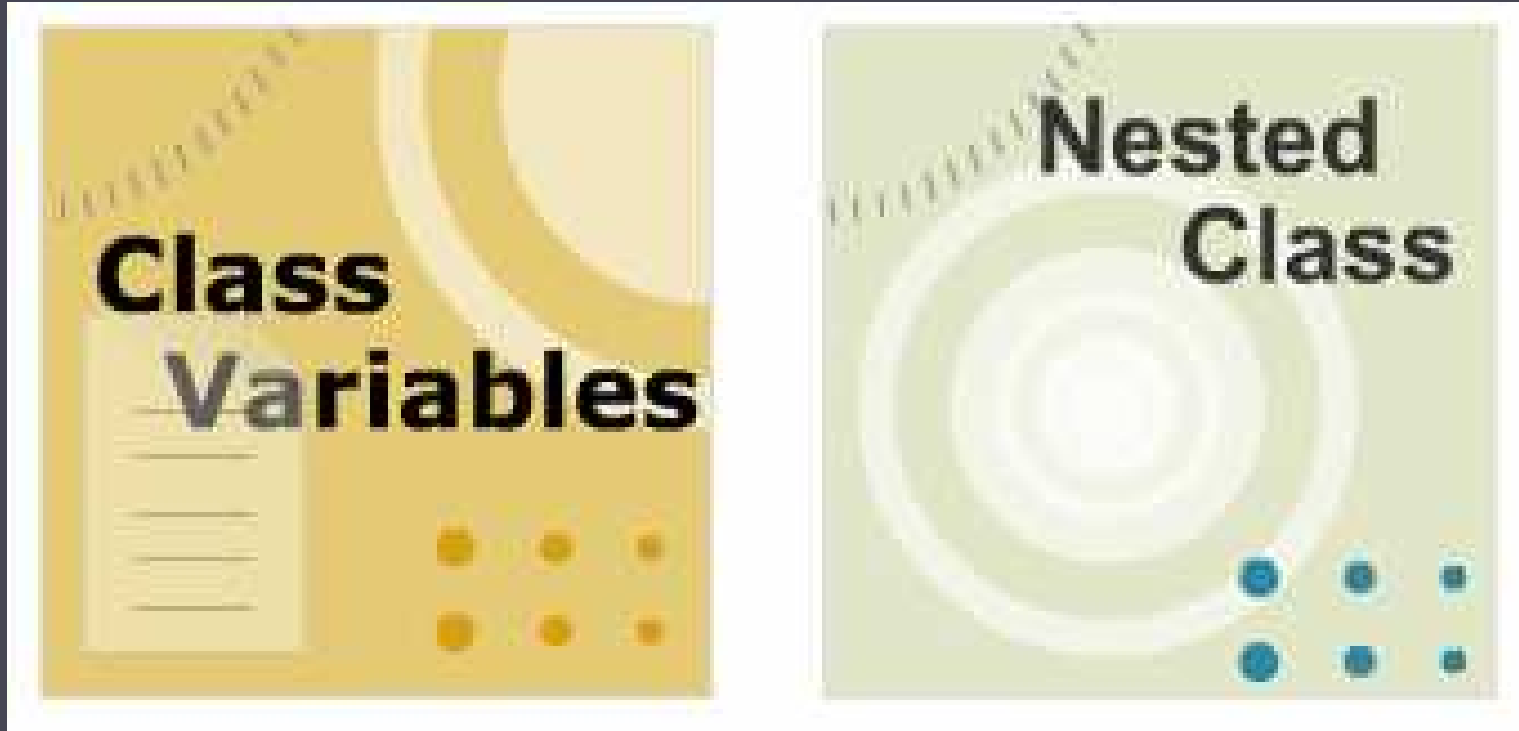
Module 6 - Packages & Access Specifiers



Module 7 - Inheritance and Interface



Module 8 - More on Classes



Module 9 - Exceptions

