9. Exceptions

8 Oct 2015

Objectives

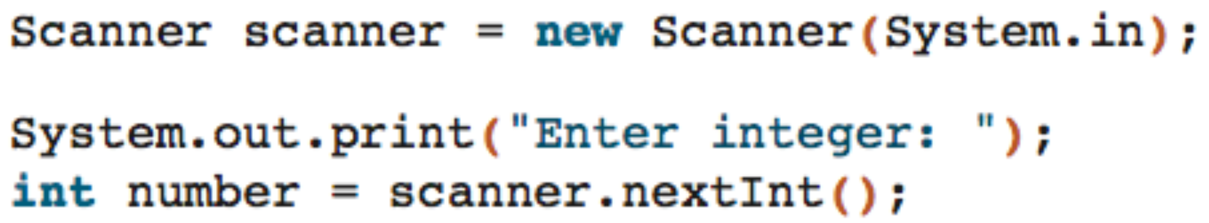
- Improve the reliability of code by incorporating exception-handling and assertion mechanisms.
- Write methods that propagate exceptions.
- Implement the try-catch blocks for catching and handling the thrown exceptions.
- Write programmer-defined exception classes.
- Distinguish between the checked and unchecked, or runtime, exceptions.

Exceptions

- An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
- When an error occurs within a method, the method creates an exception object, contains information about the error, including its type and the state of the program when the error occurred.

Exceptions

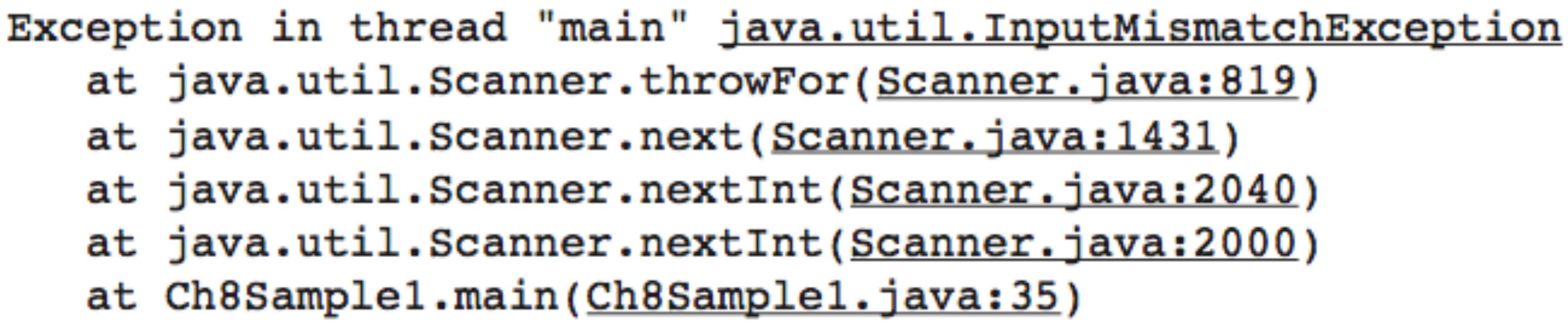
- Ex. When we use scanner.nextInt() to read int value from user



```
Scanner scanner = new Scanner(System.in);

System.out.print("Enter integer: ");
int number = scanner.nextInt();
```

Exceptions

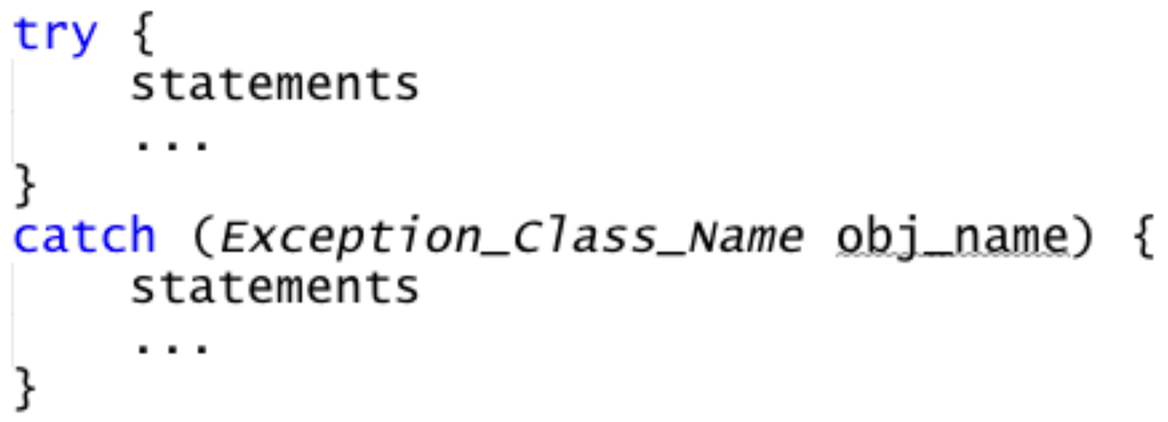
– An InputMismatchException occurs when user entered “abc123”, the nextInt() method can not convert to int.



```
Exception in thread "main" java.util.InputMismatchException
	at java.util.Scanner.throwFor(Scanner.java:819)
	at java.util.Scanner.next(Scanner.java:1431)
	at java.util.Scanner.nextInt(Scanner.java:2040)
	at java.util.Scanner.nextInt(Scanner.java:2000)
	at Ch8Sample1.main(Ch8Sample1.java:35)
```

Catching Exceptions

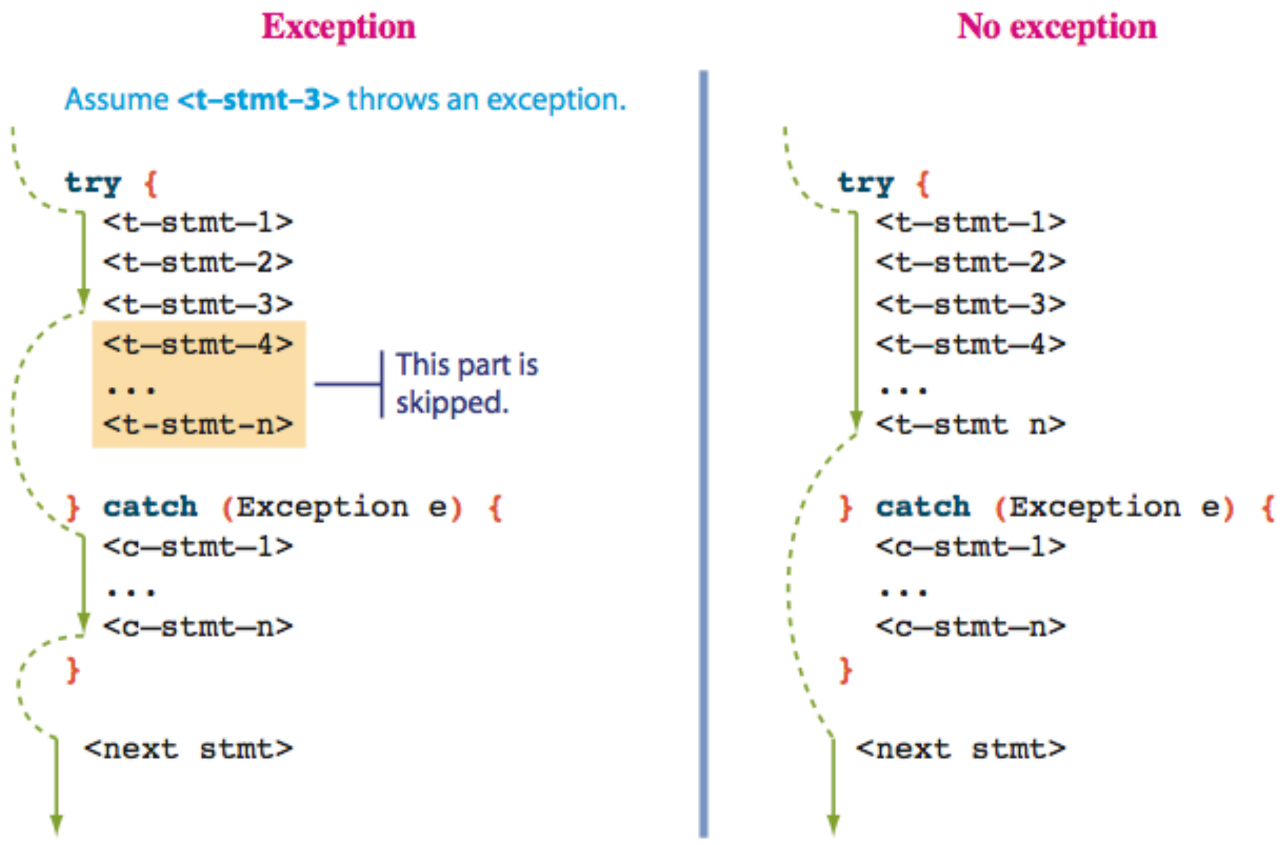
- We can handle all type of exception using try-catch block.



```
try {
      statements
      ...
}
catch (Exception_Class_Name obj_name) {
      statements
      ...
}
```

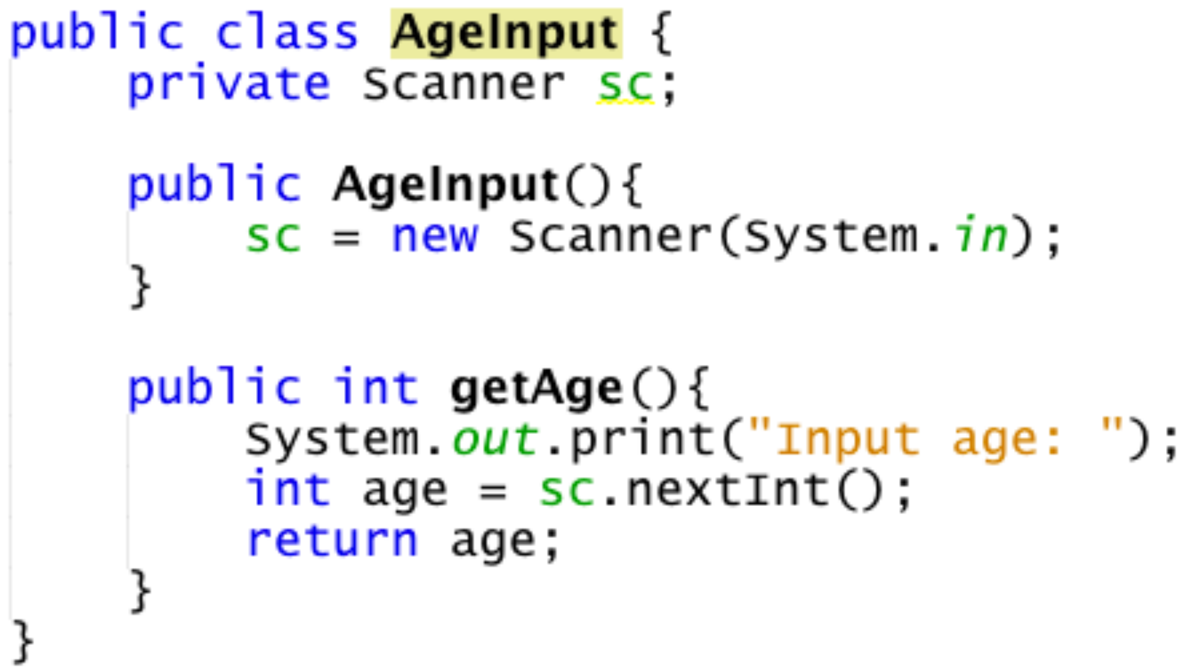
Catching Exceptions

- Control flows of the try-catch statement.



Catching Exceptions

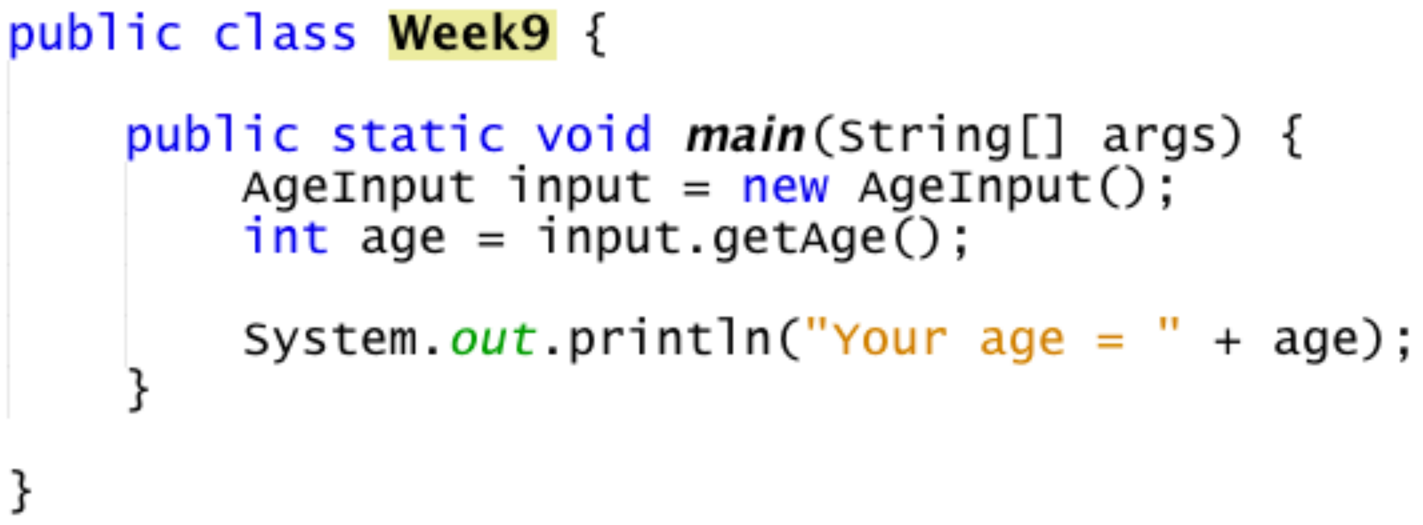
- Ex. We use AgeInput class to get age from user



```
public class AgeInput {
    private Scanner sc;

    public AgeInput(){
        sc = new Scanner(System.in);
    }

    public int getAge(){
        System.out.print("Input age: ");
        int age = sc.nextInt();
        return age;
    }
}
```

Catching Exceptions

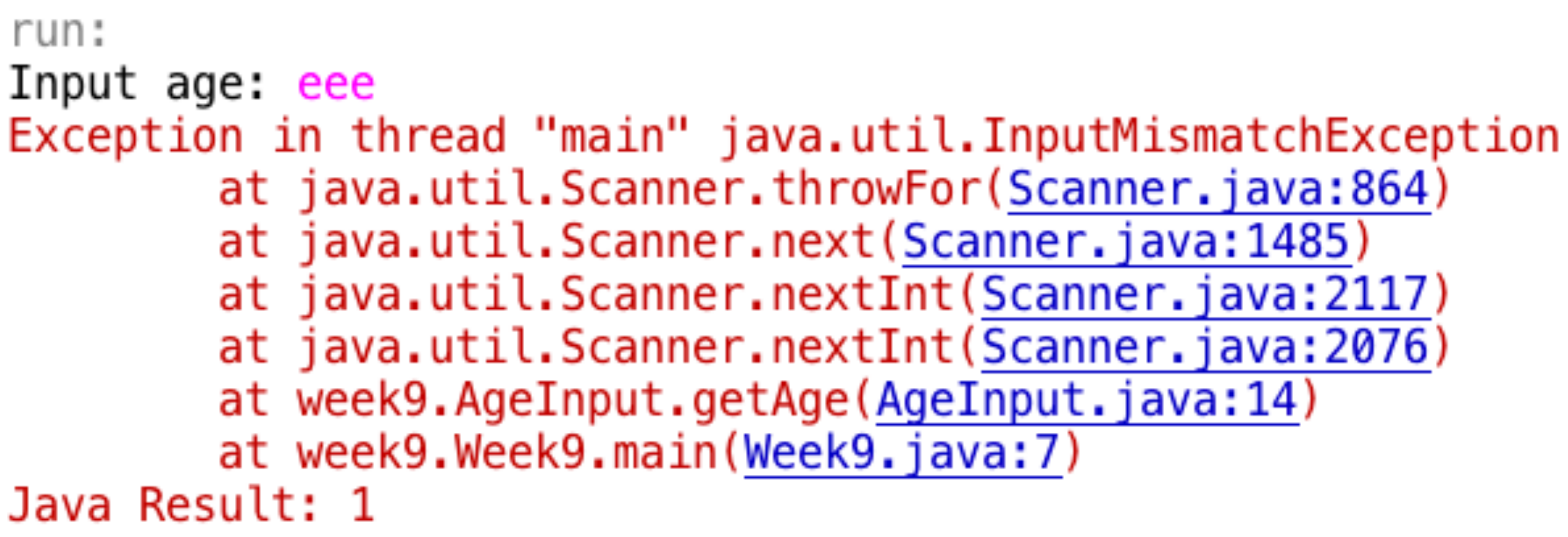
– main() method



```
public class Week9 {

    public static void main(String[] args) {
        AgeInput input = new AgeInput();
        int age = input.getAge();

        System.out.println("Your age = " + age);
    }

}
```

Catching Exceptions

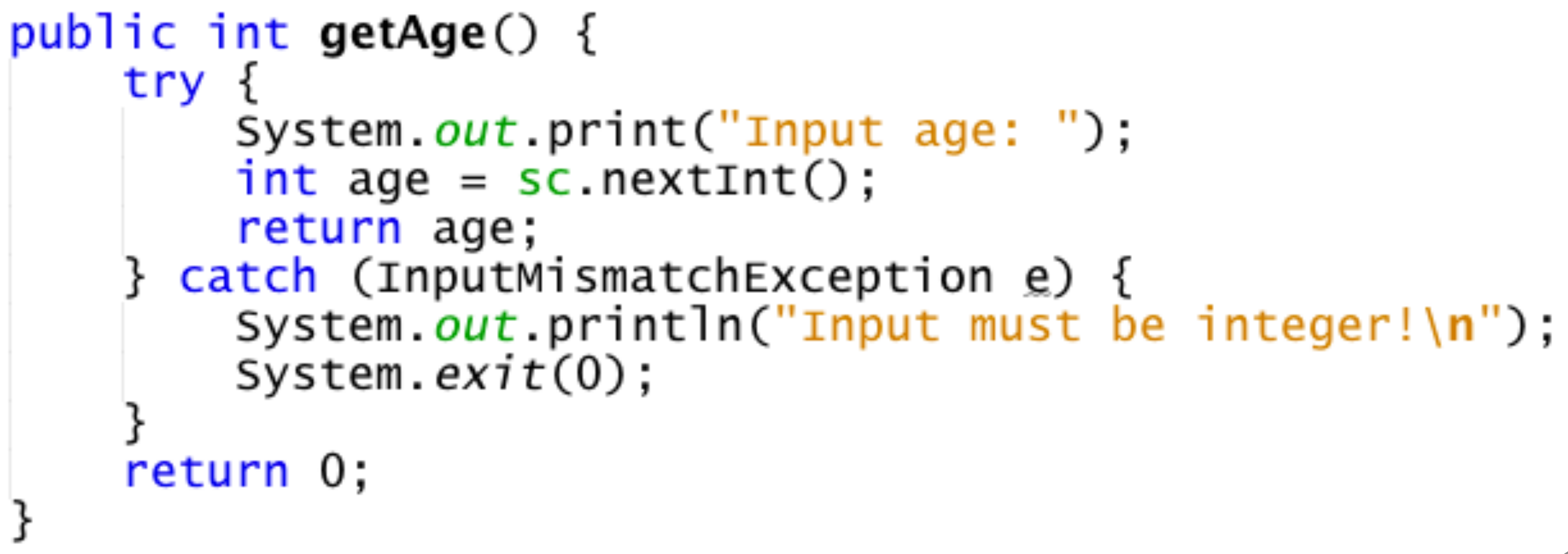
– InputMismatchException occurs when user entered “eee”



```
run:
Input age: eee
Exception in thread "main" java.util.InputMismatchException
	at java.util.Scanner.throwFor(Scanner.java:864)
	at java.util.Scanner.next(Scanner.java:1485)
	at java.util.Scanner.nextInt(Scanner.java:2117)
	at java.util.Scanner.nextInt(Scanner.java:2076)
	at week9.AgeInput.getAge(AgeInput.java:14)
	at week9.Week9.main(Week9.java:7)
Java Result: 1
```

Catching Exceptions

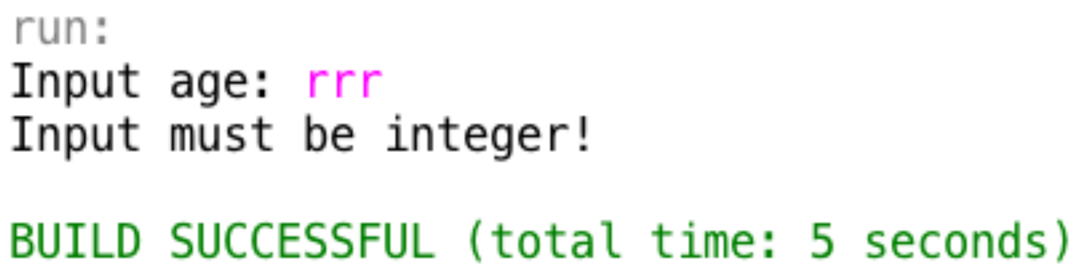
– 1st version: We can use try-catch to handle exceptions in getAge() in AgeInput class. If Exception occurs, display error message then end program.



```java
public int getAge() {
    try {
        System.out.print("Input age: ");
        int age = sc.nextInt();
        return age;
    } catch (InputMismatchException e) {
        System.out.println("Input must be integer!\n");
        System.exit(0);
    }
    return 0;
}
```

Catching Exceptions

– Output of 1st version:



```
run:
Input age: rrr
Input must be integer!

BUILD SUCCESSFUL (total time: 5 seconds)
```

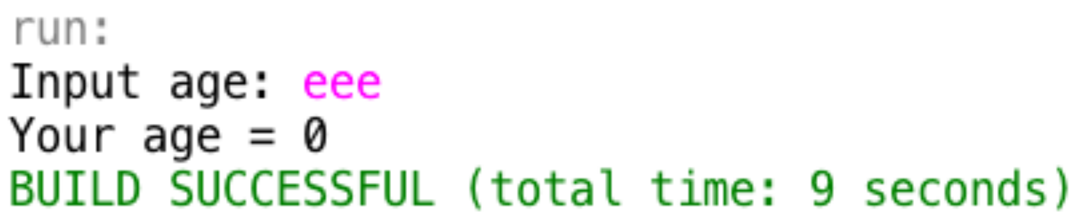
Catching Exceptions

– 2nd version: We can use try-catch to handle exceptions by set age = 0 (default value) then continue.

```java
public int getAge() {
    int age;
    try {
        System.out.print("Input age: ");
        age = sc.nextInt();
    } catch (InputMismatchException e) {
        age = 0;
    }
    return age;
}
```

Catching Exceptions

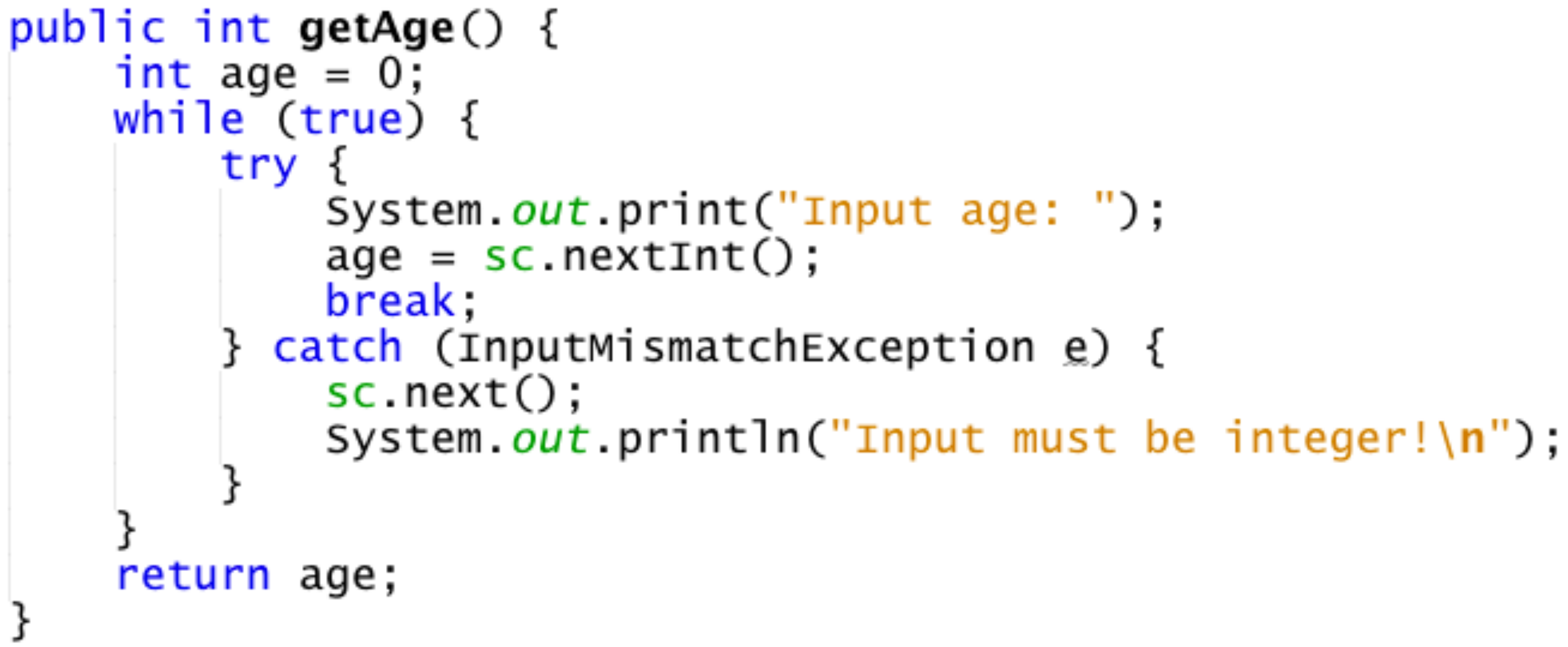
– Output of 2nd version:



```
run:
Input age: eee
Your age = 0
BUILD SUCCESSFUL (total time: 9 seconds)
```

Catching Exceptions

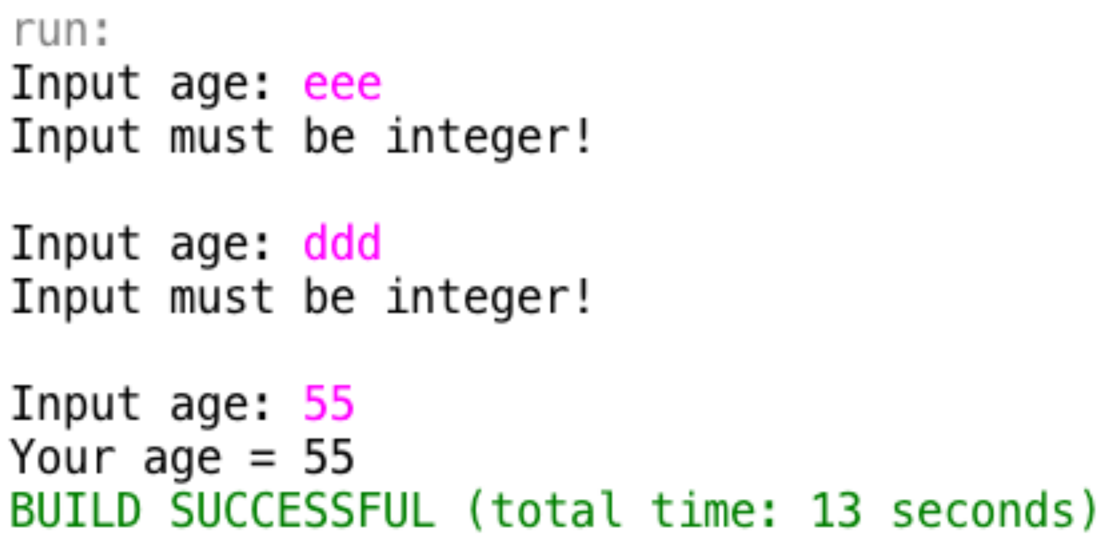
– 3rd version: Using while(true) and try-catch to handle exceptions from user input (read input value until user make valid input).



```java
public int getAge() {
    int age = 0;
    while (true) {
        try {
            System.out.print("Input age: ");
            age = sc.nextInt();
            break;
        } catch (InputMismatchException e) {
            sc.next();
            System.out.println("Input must be integer!\n");
        }
    }
    return age;
}
```

Catching Exceptions

– Output of 3rd version:



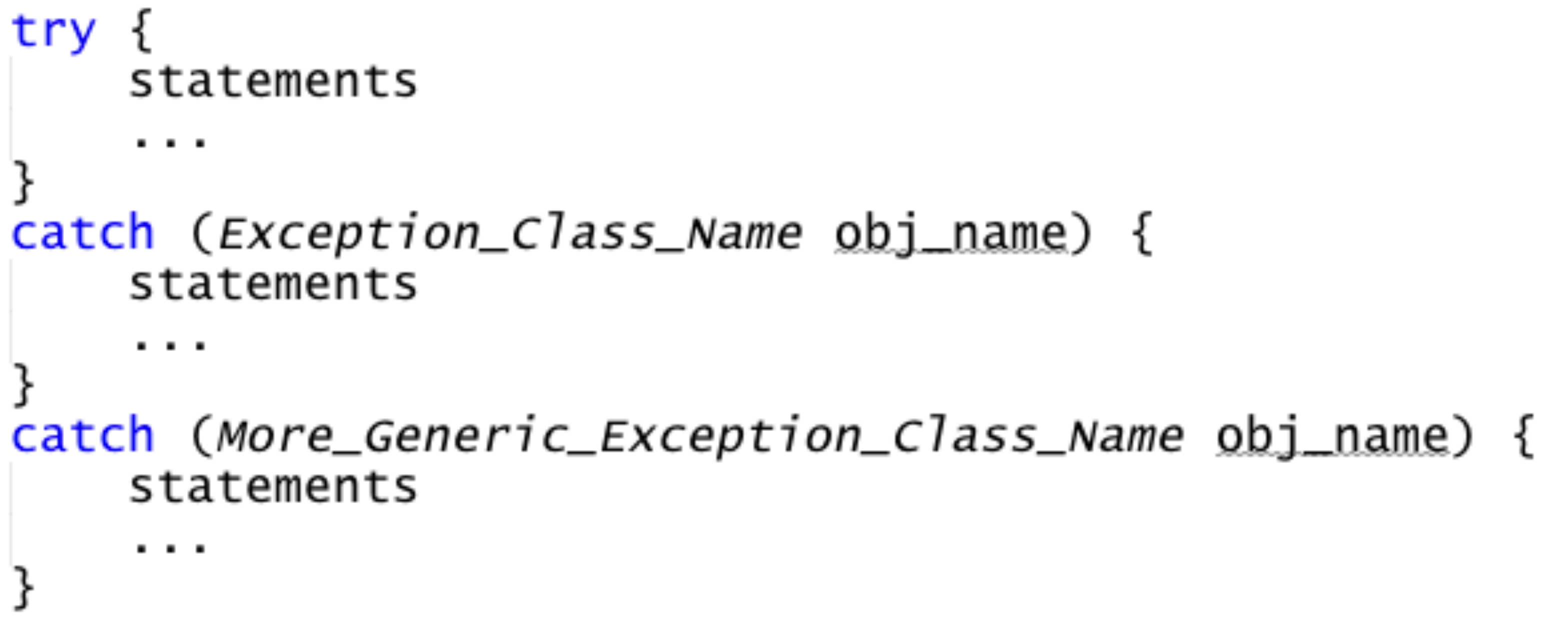
```
run:
Input age: eee
Input must be integer!

Input age: ddd
Input must be integer!

Input age: 55
Your age = 55
BUILD SUCCESSFUL (total time: 13 seconds)
```

Throwing Exceptions and Multiple catch Blocks

- In try-catch statement, we can have single or multiple catch blocks.
- When there are multiple catch blocks, it is important to check the more specialized exception classes before the more general exception classes.

Throwing Exceptions and Multiple catch Blocks

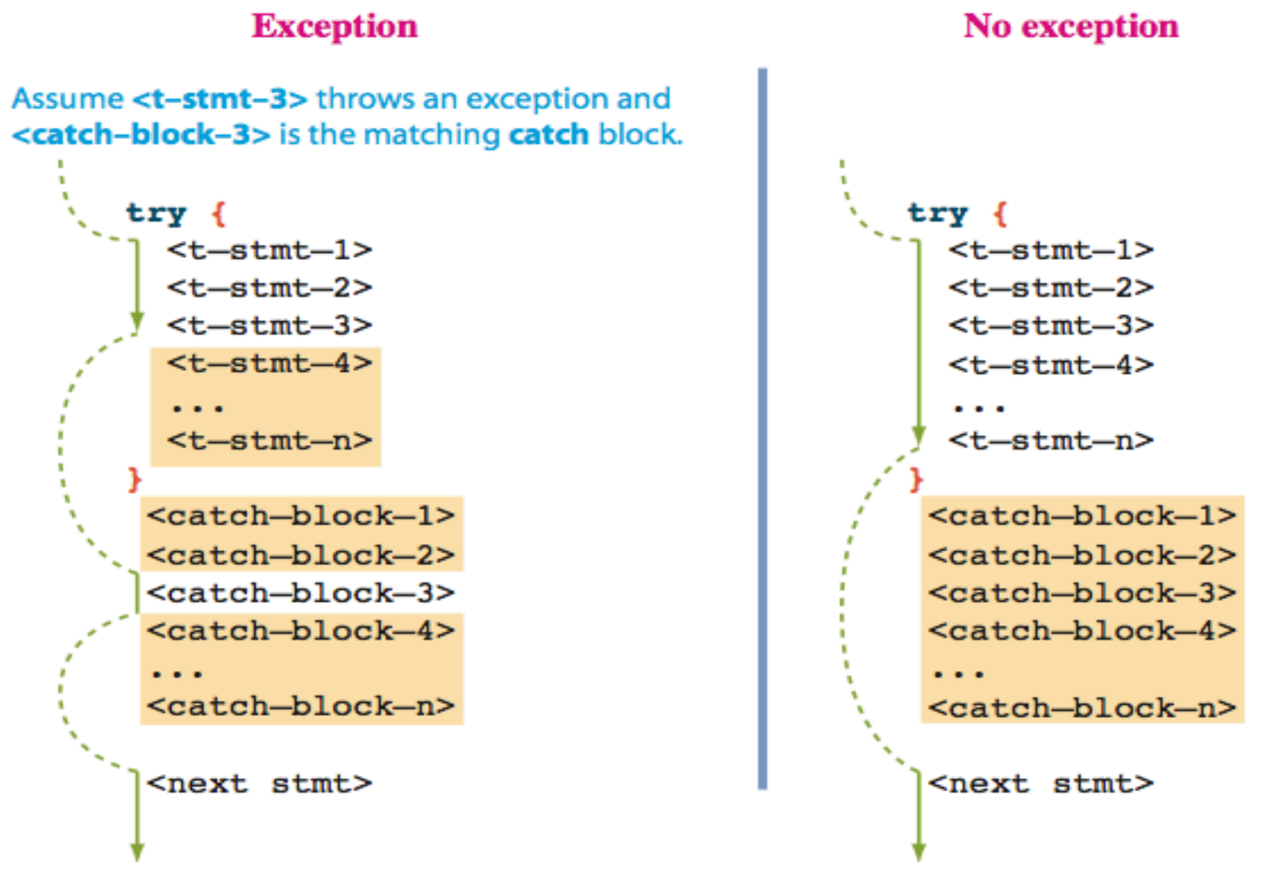
- The try-catch statement with multiple catch blocks.



```
try {
      statements
      ...
}
catch (Exception_Class_Name obj_name) {
      statements
      ...
}
catch (More_Generic_Exception_Class_Name obj_name) {
      statements
      ...
}
```

Throwing Exceptions and Multiple catch Blocks

- Control flows of the try-catch statement with multiple catch blocks.


Exception
No exception
Assume <t–stmt–3> throws an exception and <catch–block–3> is the matching catch block.

```
try {
  <t–stmt–1>
  <t–stmt–2>
  <t–stmt–3>
  <t–stmt–4>
  . . .
  <t–stmt–n>
}
<catch–block–1>
<catch–block–2>
<catch–block–3>
<catch–block–4>
. . .
<catch–block–n>

<next stmt>
```

Throwing Exceptions and Multiple catch Blocks

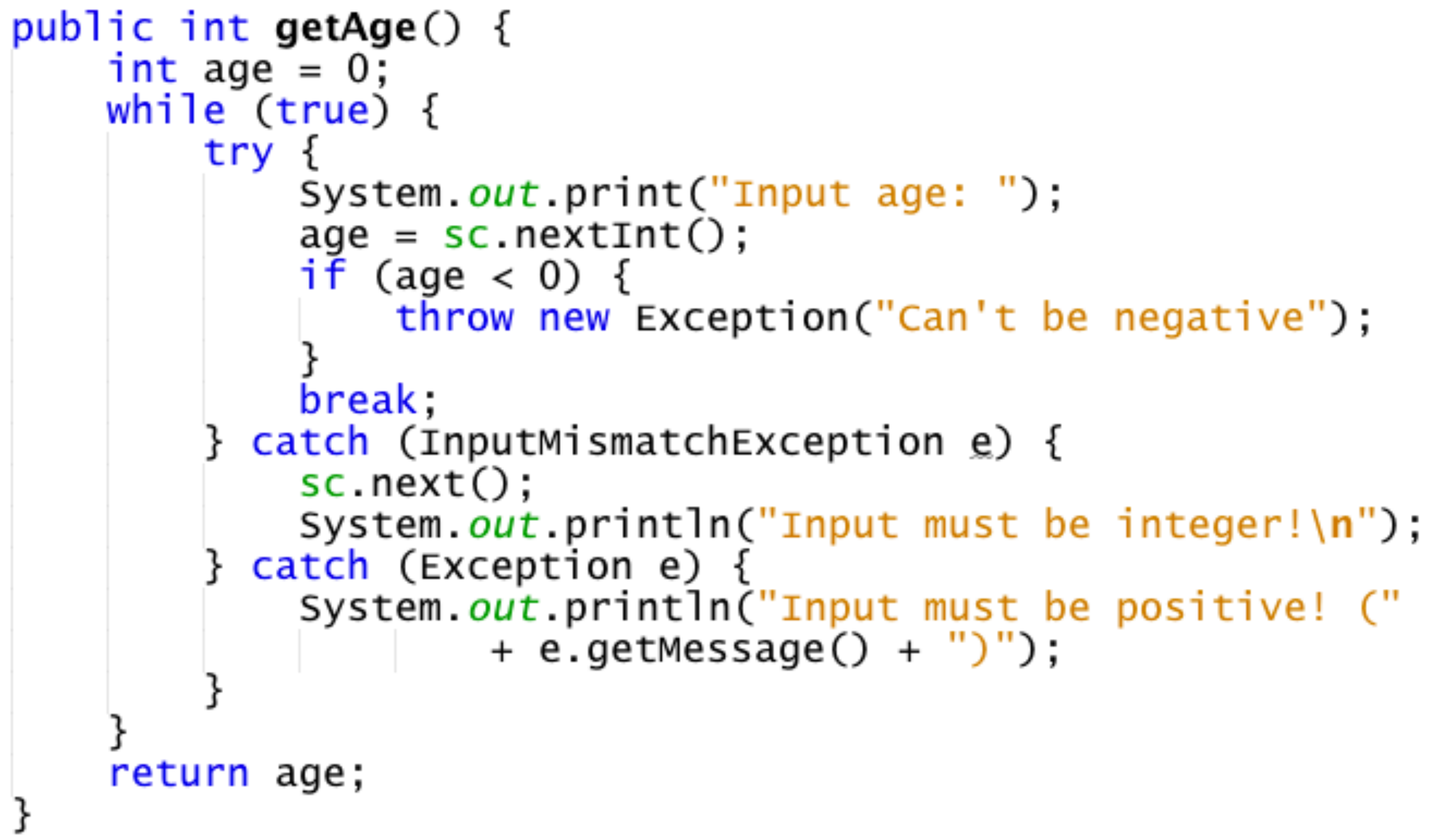
- We can throw an exception to caller by using the throw keyword.

```
throw <a throwable object>
```

```
if (num > 100) {
    throw new Exception("Out of bound");
}
```

Throwing Exceptions and Multiple catch Blocks

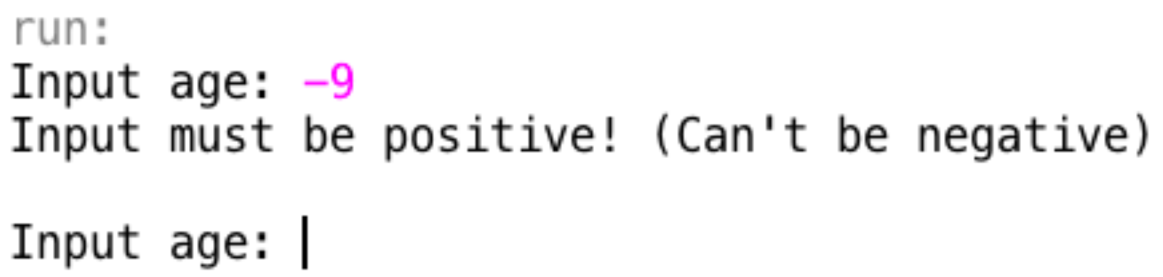
- Ex. From getAge() method, now we can use multiple catch blocks and throw an exception



```java
public int getAge() {
    int age = 0;
    while (true) {
        try {
            System.out.print("Input age: ");
            age = sc.nextInt();
            if (age < 0) {
                throw new Exception("Can't be negative");
            }
            break;
        } catch (InputMismatchException e) {
            sc.next();
            System.out.println("Input must be integer!\n");
        } catch (Exception e) {
            System.out.println("Input must be positive! ("
                    + e.getMessage() + ")");
        }
    }
    return age;
}
```

Throwing Exceptions and Multiple catch Blocks

– Output



```
run:
Input age: -9
Input must be positive! (Can't be negative)

Input age: |
```

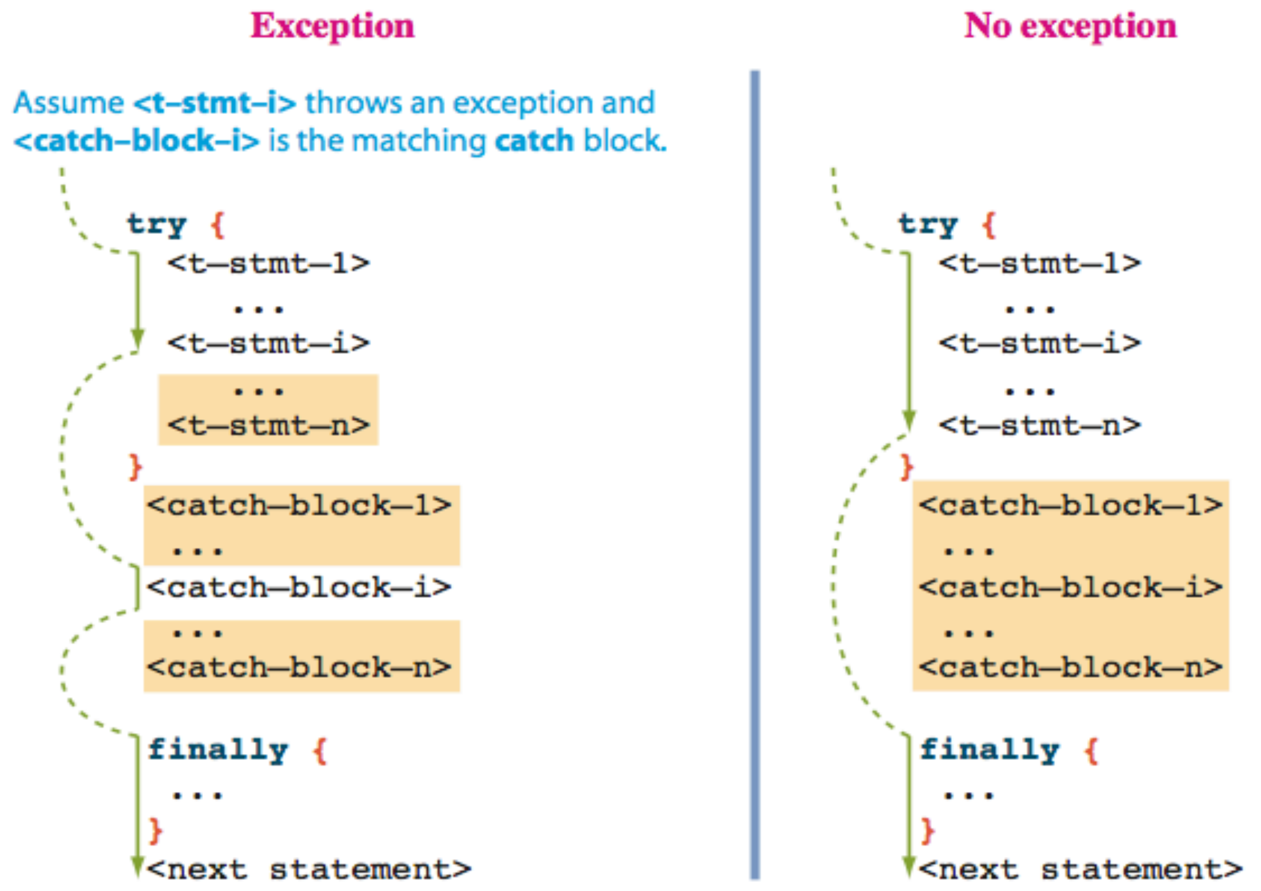
Throwing Exceptions and Multiple catch Blocks

- If there is a block of code that needs to be executed regardless of whether an exception is thrown, then we use the finally block.

```
try {
    statements
    ...
}catch (Exception_Class_Name obj_name) {
    statements
    ...
}catch (More_Generic_Exception_Class_Name obj_name) {
    statements
    ...
}finally {
    statements
    ...
}
```

Throwing Exceptions and Multiple catch Blocks

- Control flows of the try-catch-finally statement.

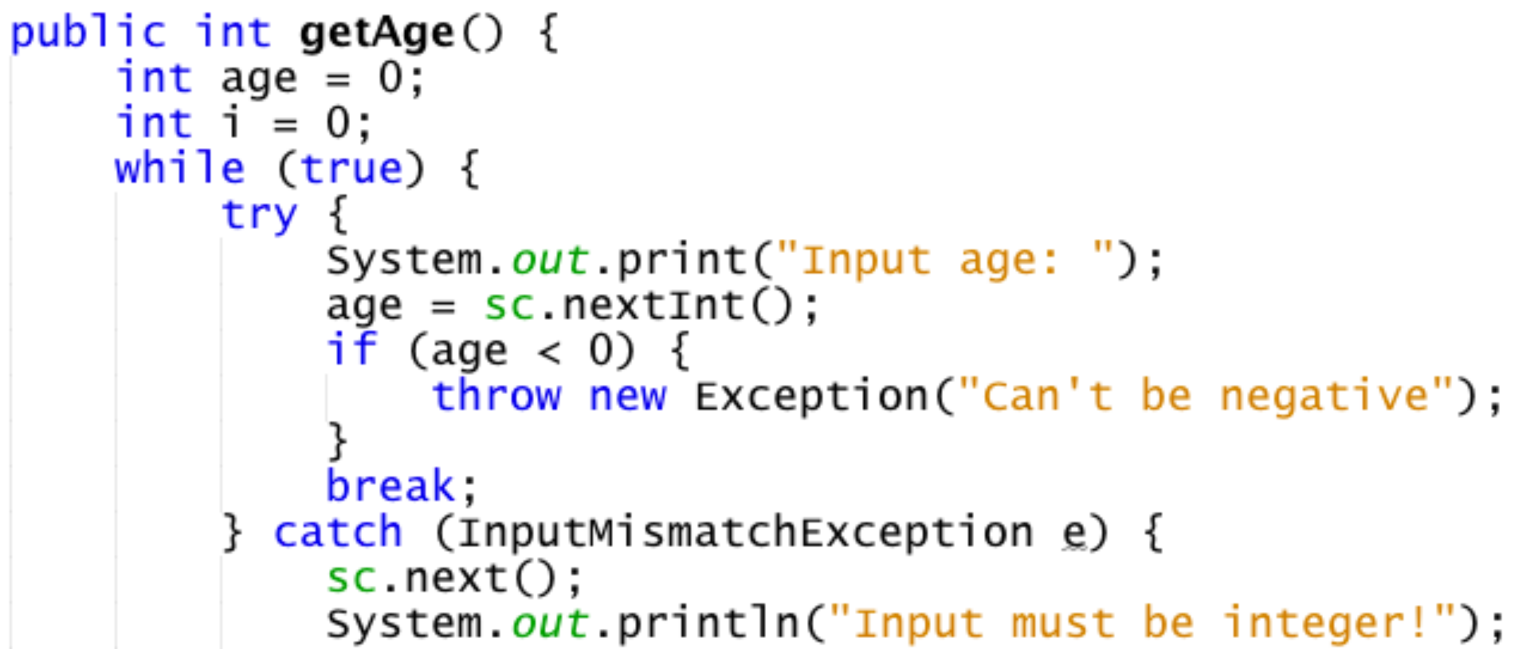


Exception
No exception
Assume <t–stmt–i> throws an exception and <catch–block–i> is the matching catch block.

```
try {
  <t–stmt–1>
     . . .
  <t–stmt–i>
     . . .
  <t–stmt–n>
}
<catch–block–1>
  . . .
<catch–block–i>
  . . .
<catch–block–n>

finally {
  . . .
}
<next statement>
```

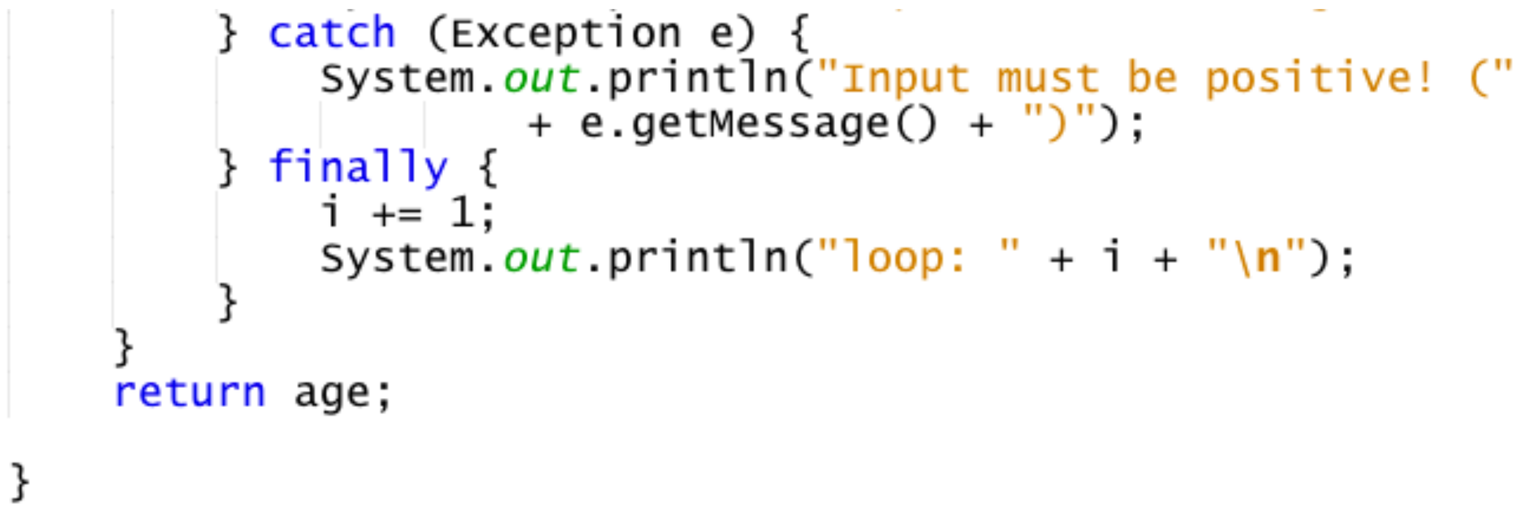
Throwing Exceptions and Multiple catch Blocks

- Ex. From getAge() method, now we use finally block to print loop count



```
public int getAge() {
    int age = 0;
    int i = 0;
    while (true) {
        try {
            System.out.print("Input age: ");
            age = sc.nextInt();
            if (age < 0) {
                throw new Exception("Can't be negative");
            }
            break;
        } catch (InputMismatchException e) {
            sc.next();
            System.out.println("Input must be integer!");
```

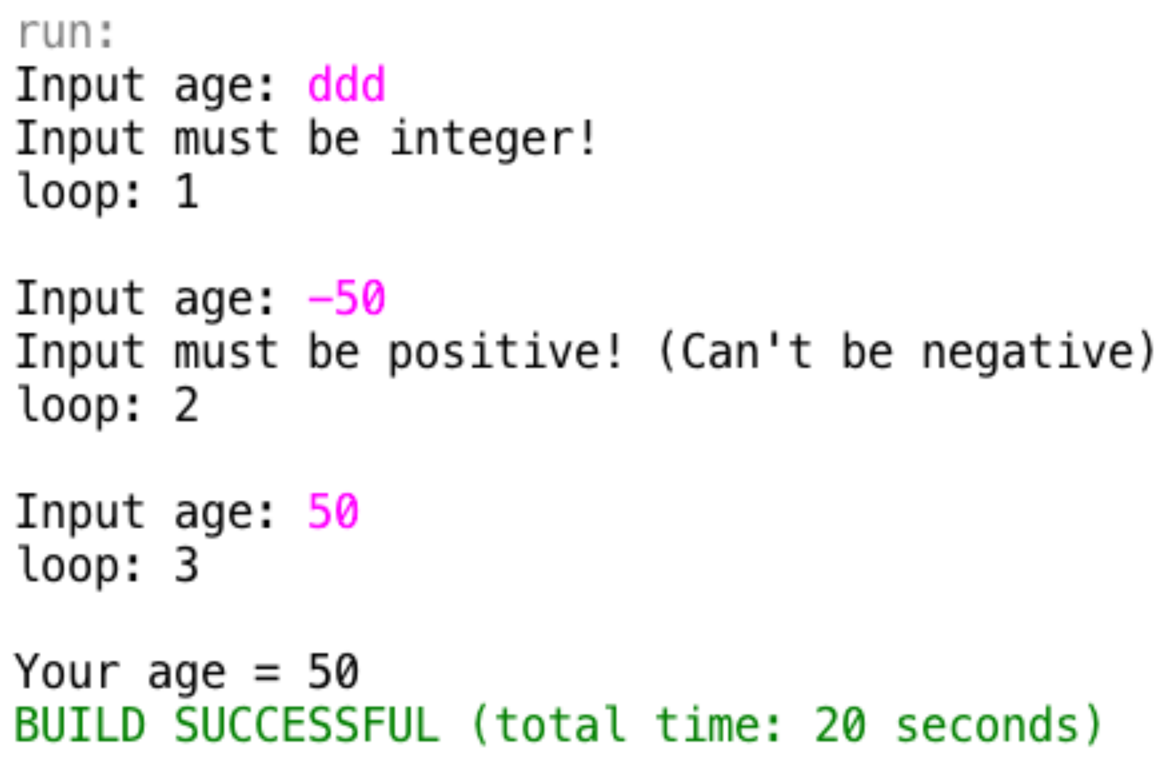
Throwing Exceptions and Multiple catch Blocks



```
    } catch (Exception e) {
      System.out.println("Input must be positive! ("
          + e.getMessage() + ")");
    } finally {
      i += 1;
      System.out.println("loop: " + i + "\n");
    }
  }
  return age;

}
```

Throwing Exceptions and Multiple catch Blocks

– Output



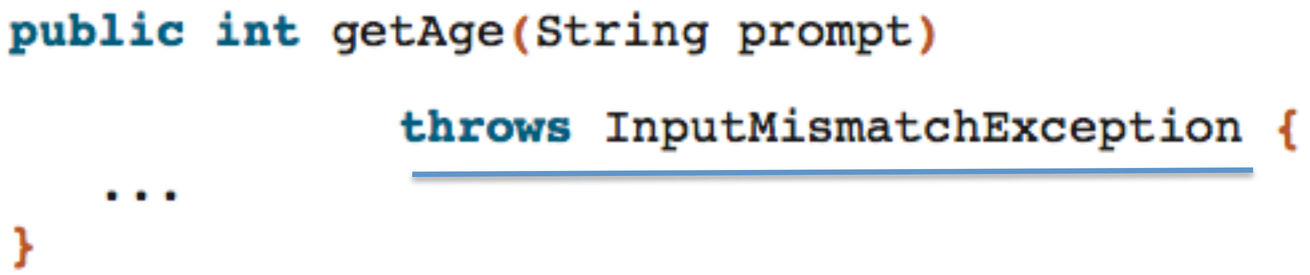
```
run:
Input age: ddd
Input must be integer!
loop: 1

Input age: -50
Input must be positive! (Can't be negative)
loop: 2

Input age: 50
loop: 3

Your age = 50
BUILD SUCCESSFUL (total time: 20 seconds)
```

Propagating Exceptions

- An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method, If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack.

Propagating Exceptions

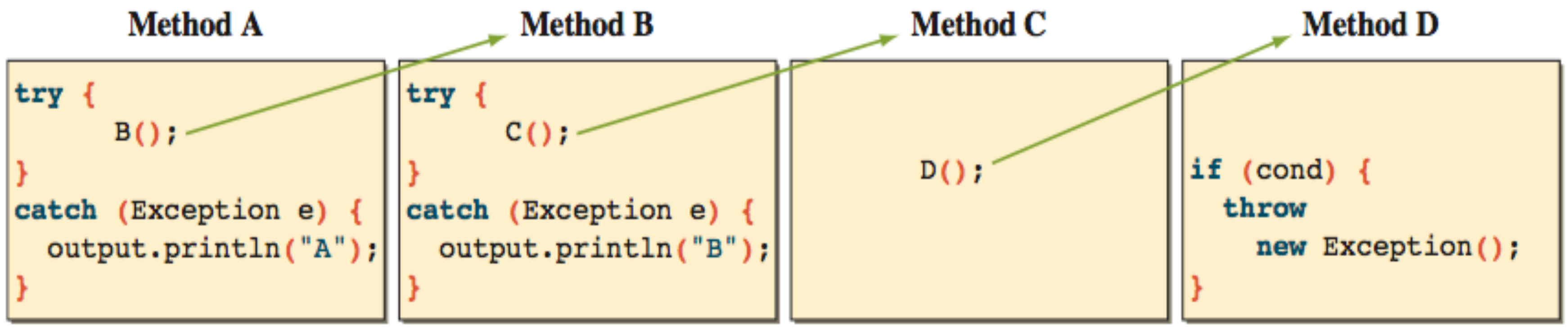
- The method that throw exception to another method must declare throws exception at method’s name



```
public int getAge(String prompt)
                throws InputMismatchException {
    ...
}
```

Propagating Exceptions

- In some class, we have methods like this



```
Method A
try {
      B();
}
catch (Exception e) {
  output.println("A");
}
```

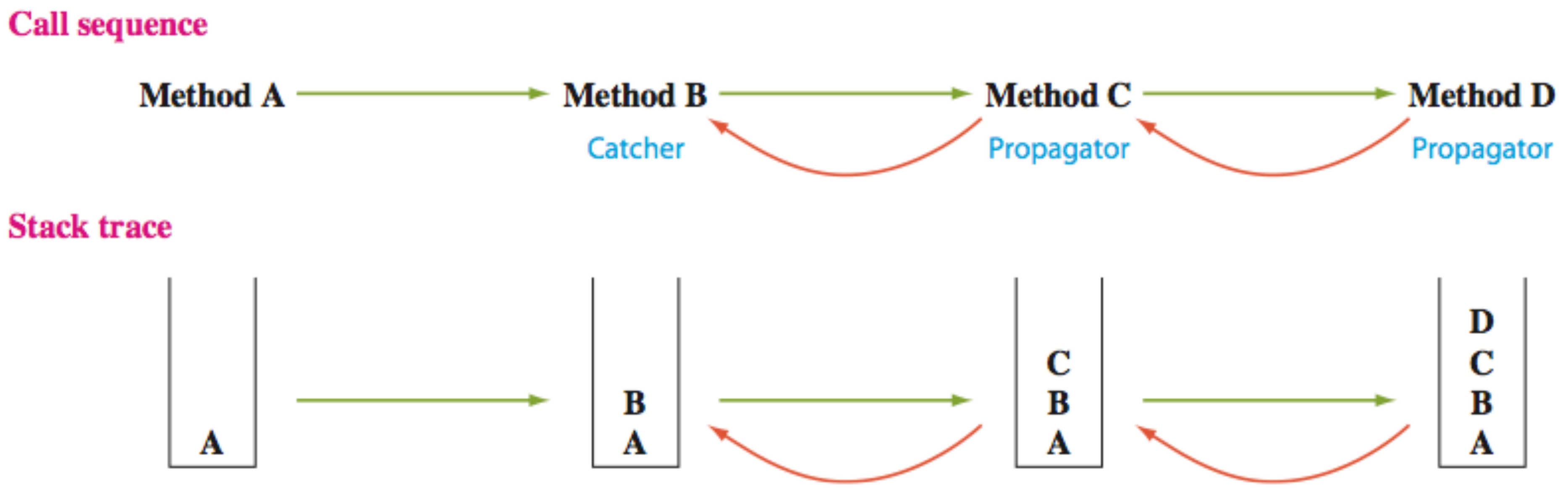
```
Method B
try {
      C();
}
catch (Exception e) {
  output.println("B");
}
```

```
Method C
        D();
```

```
Method D
if (cond) {
  throw
    new Exception();
}
```

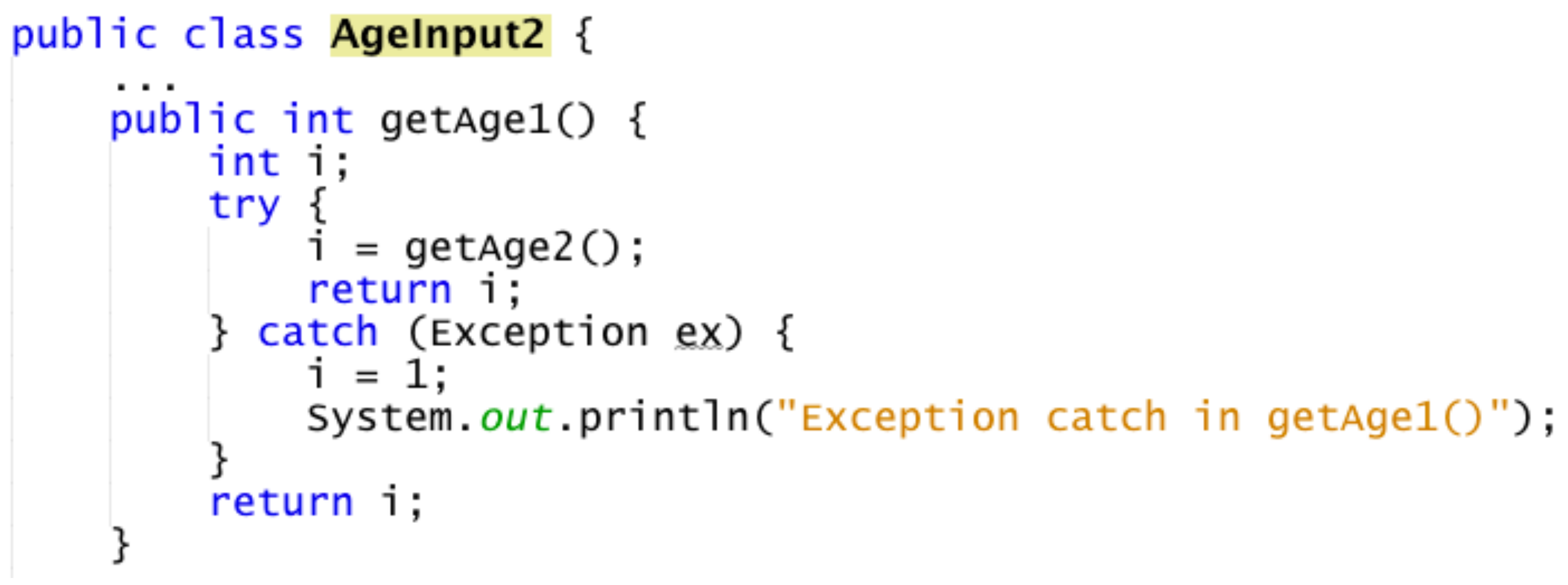
Propagating Exceptions

– From previous methods, exception first occurs in D() then it throws to C(), in C() does not catch any exception then it throws to B(), in B() exception is caught in catch block



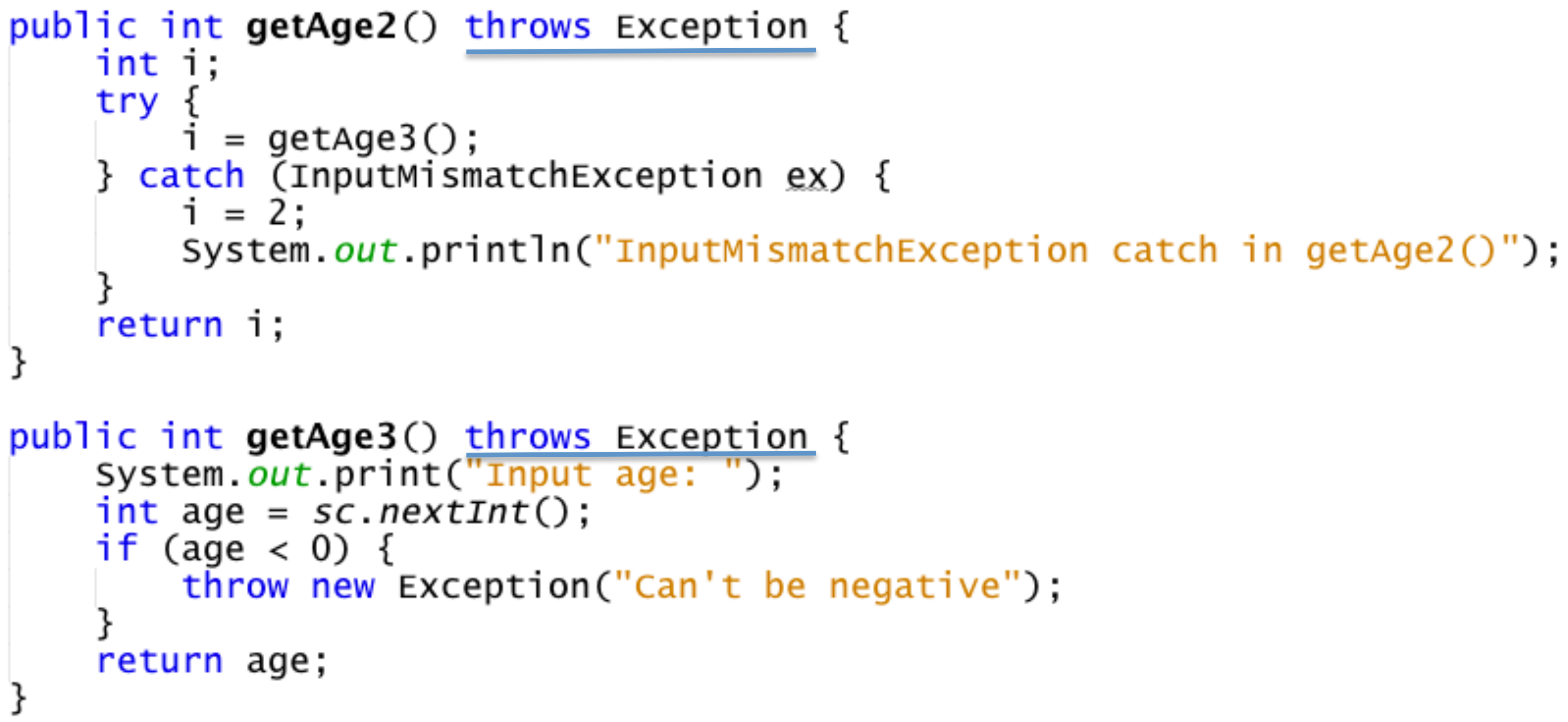
Propagating Exceptions

- Ex. In AgeInput2 class, there are getAge1(), getAge2(), getAge3()



```
public class AgeInput2 {
    ...
    public int getAge1() {
        int i;
        try {
            i = getAge2();
            return i;
        } catch (Exception ex) {
            i = 1;
            System.out.println("Exception catch in getAge1()");
        }
        return i;
    }
```

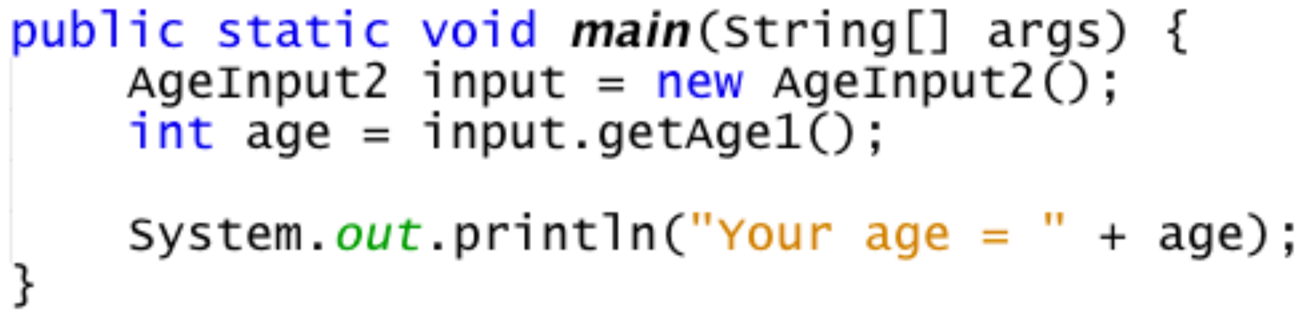
Propagating Exceptions



```
public int getAge2() throws Exception {
    int i;
    try {
        i = getAge3();
    } catch (InputMismatchException ex) {
        i = 2;
        System.out.println("InputMismatchException catch in getAge2()");
    }
    return i;
}

public int getAge3() throws Exception {
    System.out.print("Input age: ");
    int age = sc.nextInt();
    if (age < 0) {
        throw new Exception("Can't be negative");
    }
    return age;
}
```

Propagating Exceptions

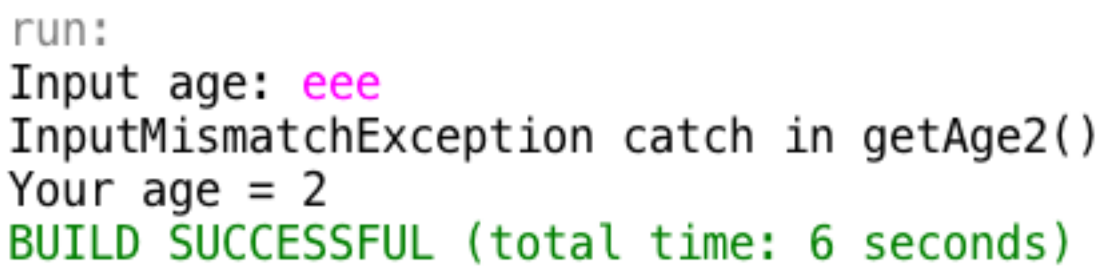
– main() method



```
public static void main(String[] args) {
    AgeInput2 input = new AgeInput2();
    int age = input.getAge1();

    System.out.println("Your age = " + age);
}
```

Propagating Exceptions

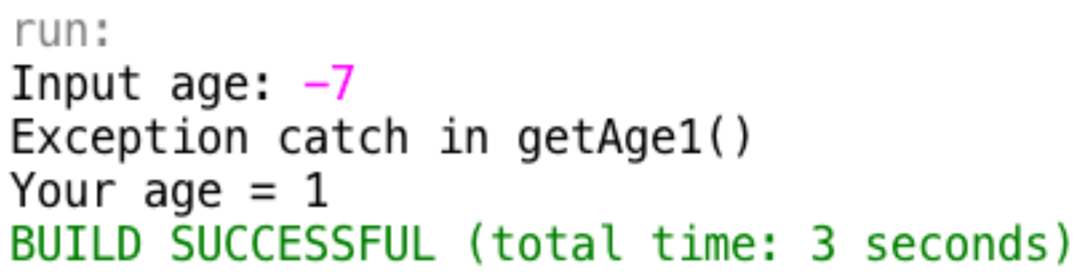
- Output: If user entered “eee”, InputMismatchException occurs in getAge3() then it is thrown to getAge2() and caught by catch block



```
run:
Input age: eee
InputMismatchException catch in getAge2()
Your age = 2
BUILD SUCCESSFUL (total time: 6 seconds)
```

Propagating Exceptions

– Output: If user entered “-7”, Exception occurs in getAge3() then it is thrown to getAge1() and caught by catch block



```
run:
Input age: -7
Exception catch in getAge1()
Your age = 1
BUILD SUCCESSFUL (total time: 3 seconds)
```

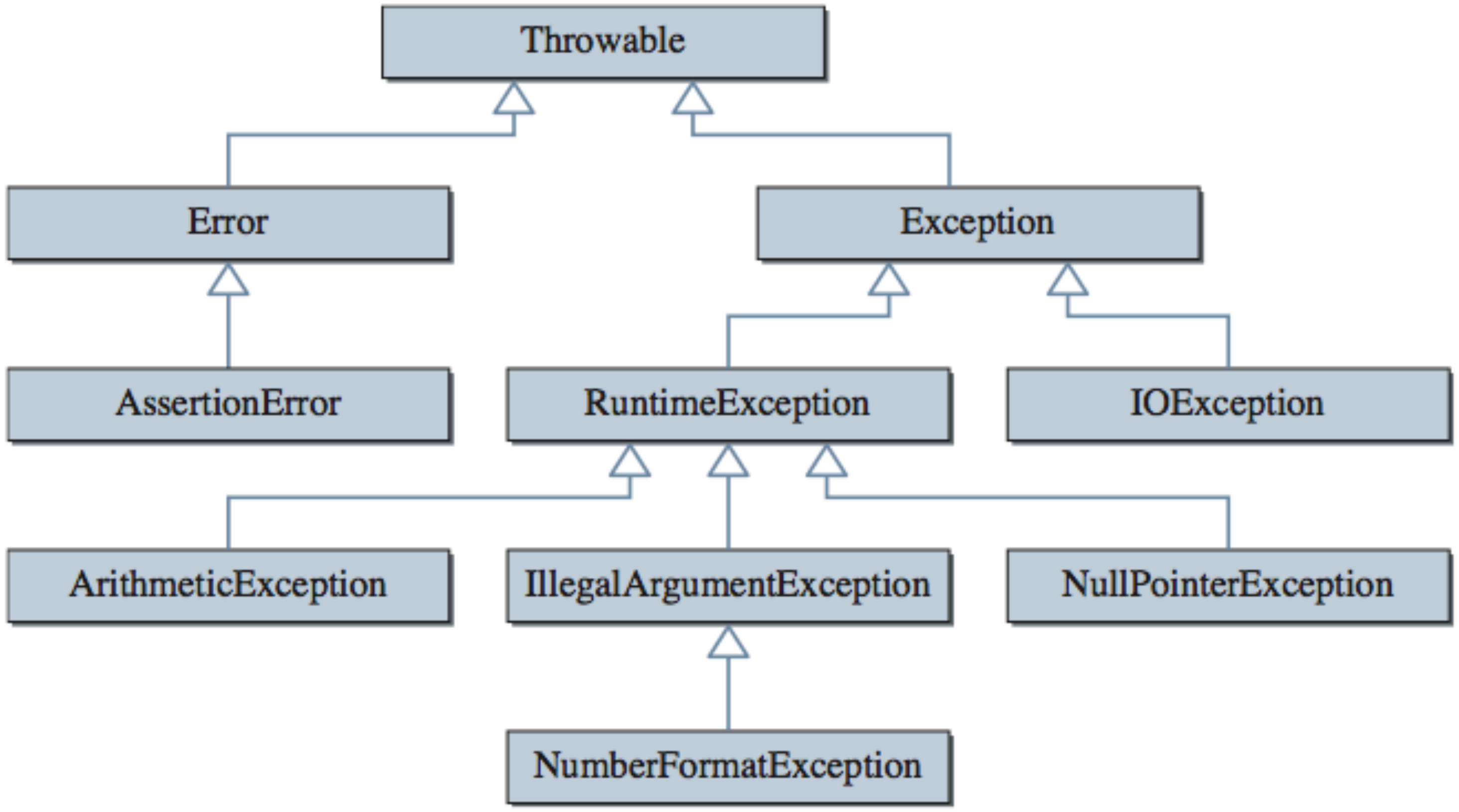
Types of Exceptions

- There are 2 types of Exception
  1. Checked Exception: is an exception that is checked at compile time eg.
     - FileNotFoundException

  2. Unchecked Exception (Runtime Exception): is an exception that is unchecked at compile time eg.
     - ArithmeticException
     - InputMismatchException

Types of Exceptions

- Some classes in the inheritance hierarchy from the Throwable class.

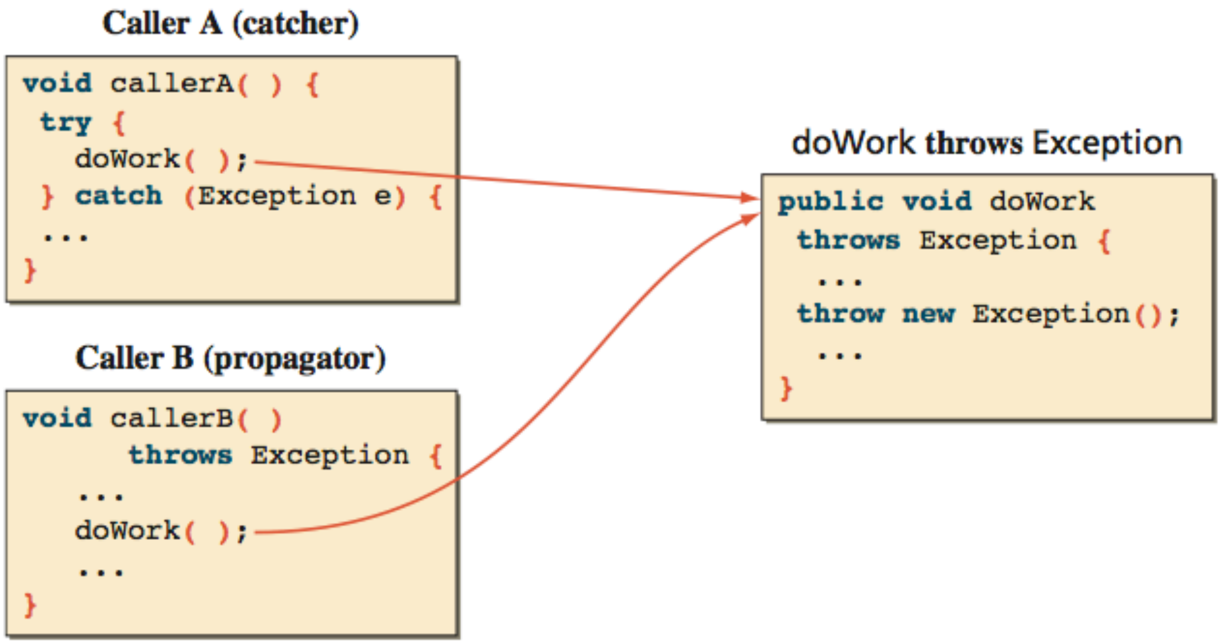


Types of Exceptions

- Callers of a method that can throw a checked exception must explicitly include the try-catch statement in the method body or the throws clause in the method header.

Types of Exceptions

– Callers of a method that can throw a checked exception.



```
void callerA( ) {
 try {
   doWork( );
 } catch (Exception e) {
 ...
}
```

```
void callerB( )
       throws Exception {
   ...
   doWork( );
   ...
}
```

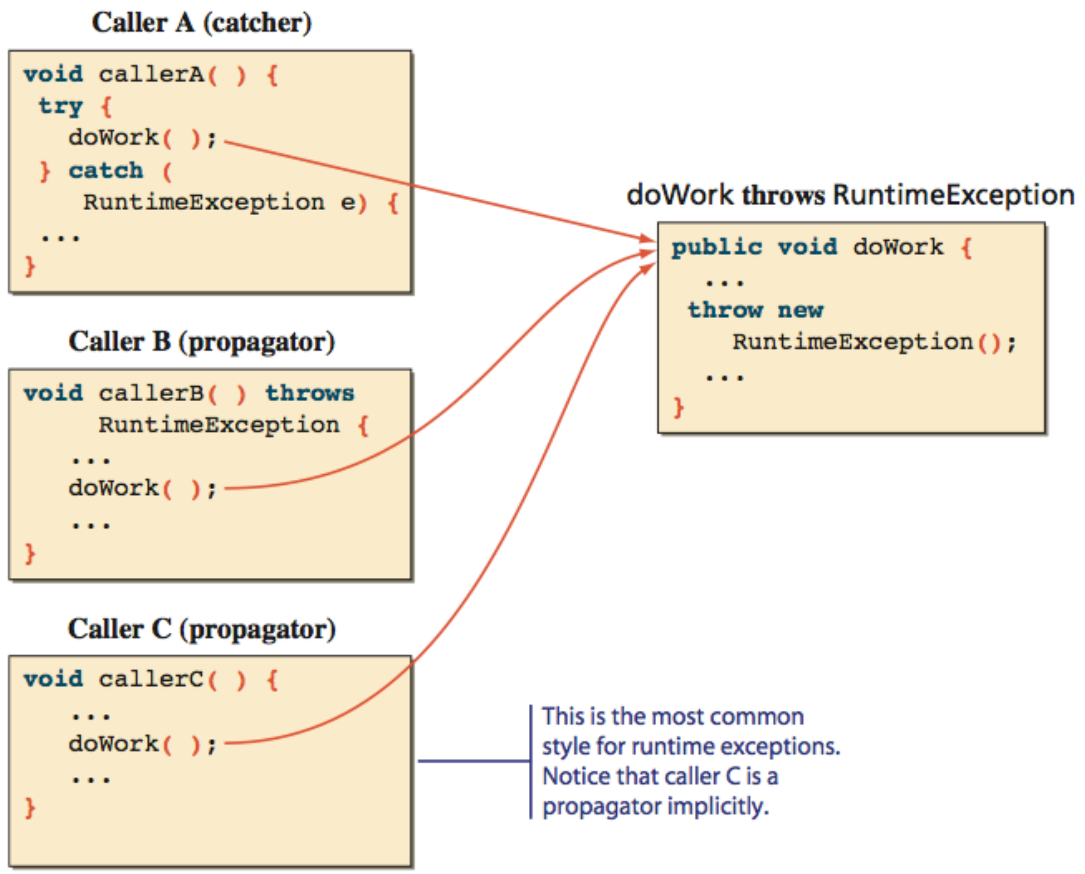
```
public void doWork
 throws Exception {
  ...
 throw new Exception();
  ...
}
```

Types of Exceptions

- Callers of a method that can throw runtime or unchecked exceptions, it is optional to include the try-catch statement or the throws clause in the method header.

Types of Exceptions

– Callers of a method that can throw runtime or unchecked exceptions.


Caller A (catcher)
Caller B (propagator)
Caller C (propagator)
doWork throws RuntimeException
This is the most common style for runtime exceptions. Notice that caller C is a propagator implicitly.

```
void callerA( ) {
 try {
   doWork( );
 } catch (
    RuntimeException e) {
 ...
}
```

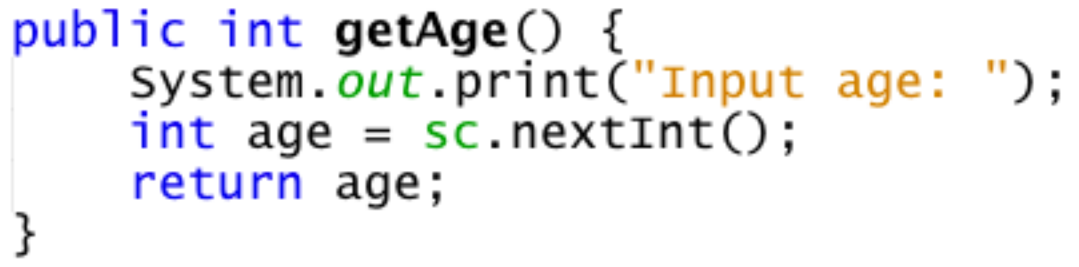
```
void callerB( ) throws
     RuntimeException {
   ...
   doWork( );
   ...
}
```

```
void callerC( ) {
   ...
   doWork( );
   ...
}
```

```
public void doWork {
   ...
 throw new
    RuntimeException();
   ...
}
```

Types of Exceptions

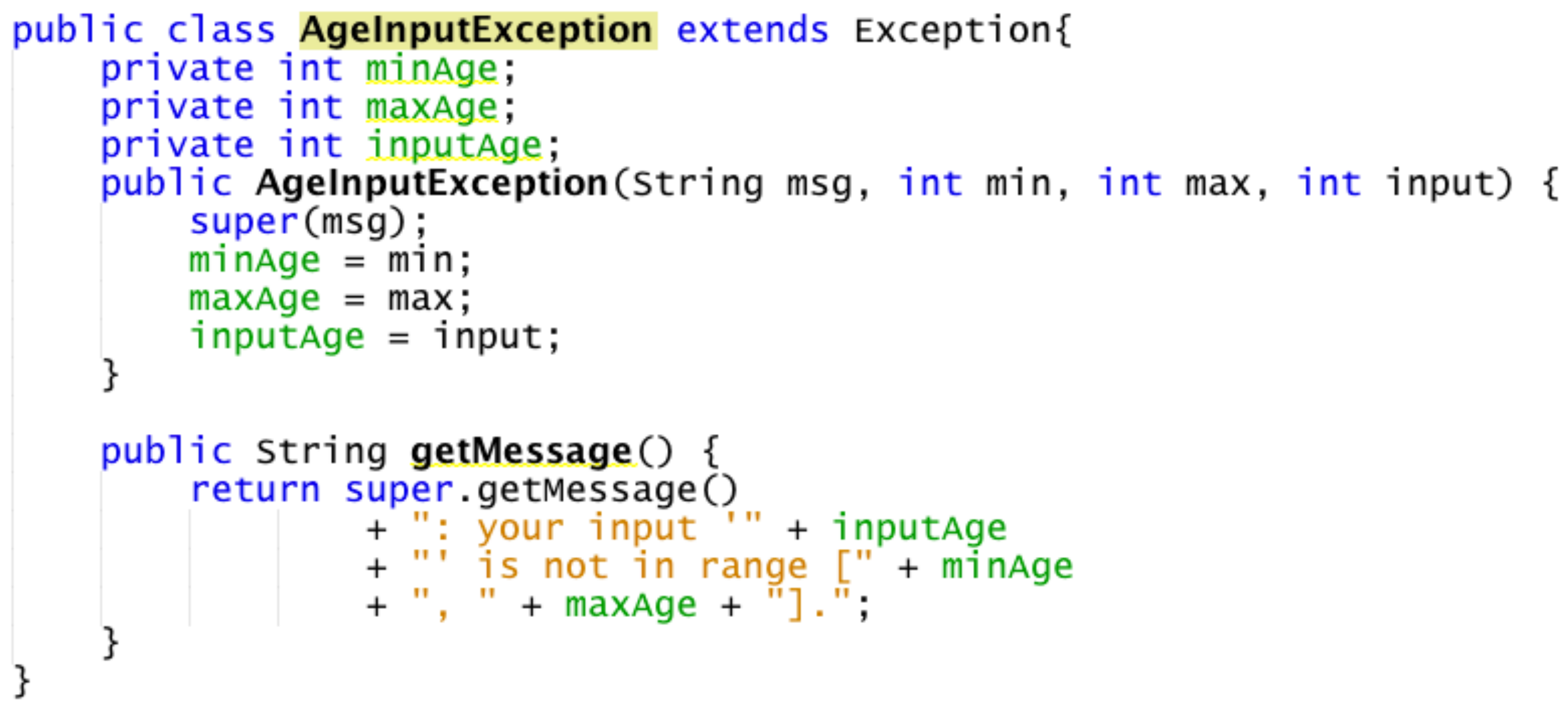
- Ex. InputMismatchException is RuntimeException then it is optional to include the try-catch statement or the throws clause in the method header.



```
public int getAge() {
    System.out.print("Input age: ");
    int age = sc.nextInt();
    return age;
}
```

Programmer-Defined Exceptions

- We can define our own exception classes by extends Exception.
  - AgeInputException class



```java
public class AgeInputException extends Exception{
    private int minAge;
    private int maxAge;
    private int inputAge;
    public AgeInputException(String msg, int min, int max, int input) {
        super(msg);
        minAge = min;
        maxAge = max;
        inputAge = input;
    }

    public String getMessage() {
        return super.getMessage()
                + ": your input '" + inputAge
                + "' is not in range [" + minAge
                + ", " + maxAge + "].";
    }
}
```

Programmer-Defined Exceptions

– AgeInput class

```java
public class AgeInput {
    private final int MIN_AGE = 1;
    private final int MAX_AGE = 100;
    private Scanner sc;

    public AgeInput() {
        sc = new Scanner(System.in);
    }

    public int getAge() throws AgeInputException{
        System.out.print("Input age: ");
        int age = sc.nextInt();
        if(age < MIN_AGE || age > MAX_AGE){
            throw new AgeInputException("AgeInputException occurs",
                    MIN_AGE, MAX_AGE, age);
        }
        return age;
    }
}
```

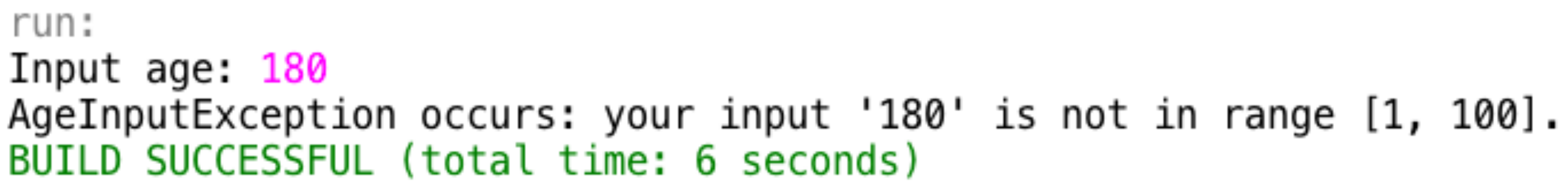
Programmer-Defined Exceptions

– main() method

```
public static void main(String[] args) {

    AgeInput input = new AgeInput();
    int age;
    try {
        age = input.getAge();
        System.out.println("Your age = " + age);
    } catch (AgeInputException ex) {
        System.out.println(ex.getMessage());
    }
}
```

Programmer-Defined Exceptions

– Output: If user entered “180”, AgeInputException occurs in getAge() method in AgeInput class then it is thrown to main() and caught by catch block then display error message.



```
run:
Input age: 180
AgeInputException occurs: your input '180' is not in range [1, 100].
BUILD SUCCESSFUL (total time: 6 seconds)
```

Summary

- Exception handling is technique to improve program reliability.
- Exception handling is another type of control flow.
- An exception represents an error condition, and when it occurs, we say an exception is thrown.
- A thrown exception must be handled by either catching it or propagating it to other methods.

Reference

- C. Thomas Wu, An Introduction to Object-Oriented Programming with Java, 5th Edition
  - Chapter 8: Exceptions and Assertions

Question?