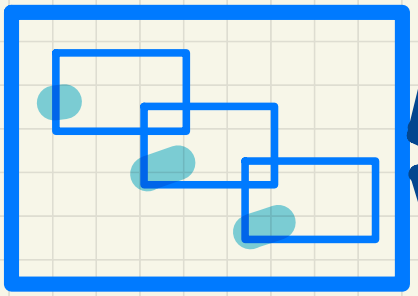
Lecture 1

Part A

Elementary Programming - Development Process

Separation of Concerns

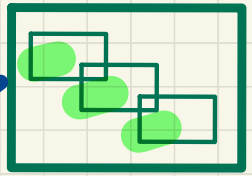


- Classes & Methods
- Methods
  * containing no print statements
  * return statements

use

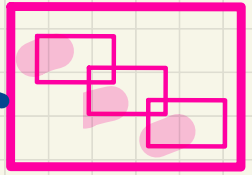
use

junit_tests



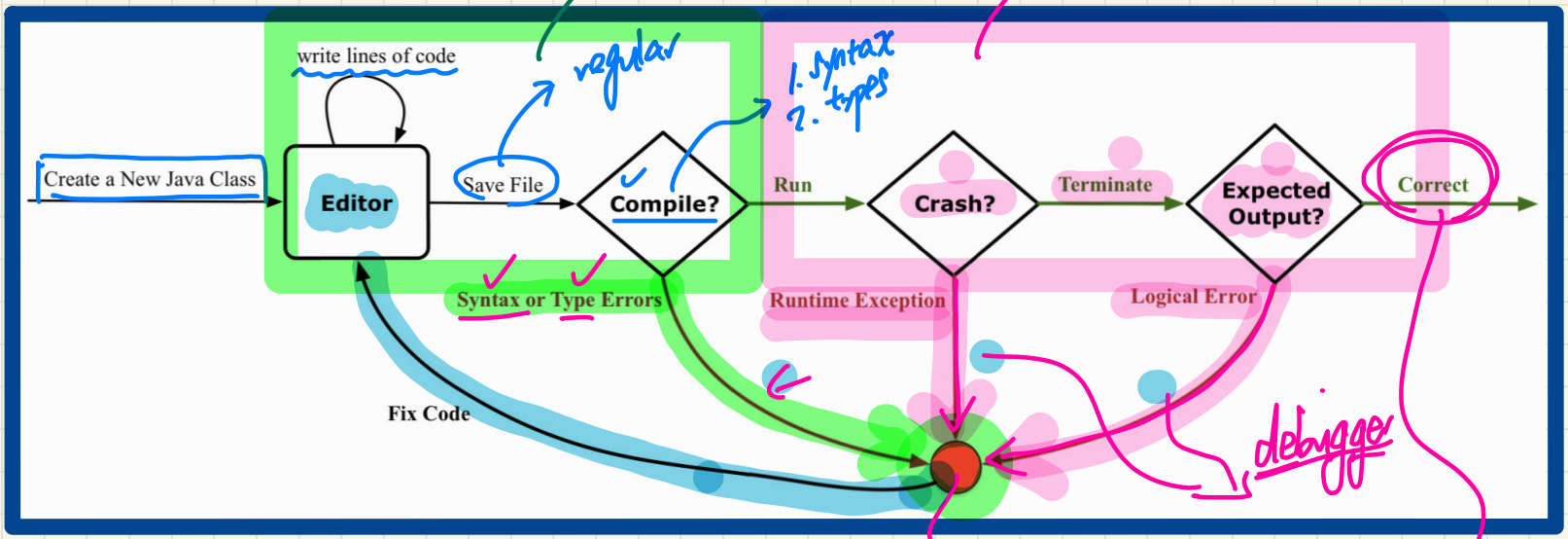
- Expected vs. Actual Values
- Methods
  * calling methods from model
  * containing no print statements
  * assertions

console_apps



- main method
  * calling methods from model
  * containing print statements
  * containg no return statements

Development Process



compile-time

run-time

regular

1. syntax
2. types

debugger

error state (something wrong).

1. compiles
2. terminates
3. output expected

Error at Compile Time: Syntax Errors (1)

CompileTimeSyntaxError1.java

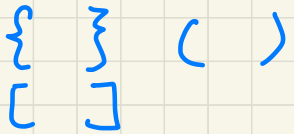
```
public class CompileTimeSyntaxError1 {
    public static void main(String[] args) {
        // Syntax Error: missing semicolon
        System.out.println("Hello") ;
    }
}
```

Error at Compile Time: Syntax Errors (2)

CompileTimeSyntaxError2.java

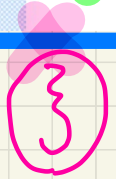
```
public class CompileTimeSyntaxError2 {
    public static void main(String[] args) {
        // Syntax Error: missing ending double quote
        System.out.println("Hello);
    }
}
```

Error at Compile Time: Syntax Errors (3)



CompileTimeSyntaxError3.java

```
public class CompileTimeSyntaxError3 {
    public static void main(String[] args) {
        System.out.println("Hello");

        /* Error 3: missing ending curly bracket */
}
```



Error at Compile Time: Syntax Errors (4)

CompileTimeSyntaxError4.java

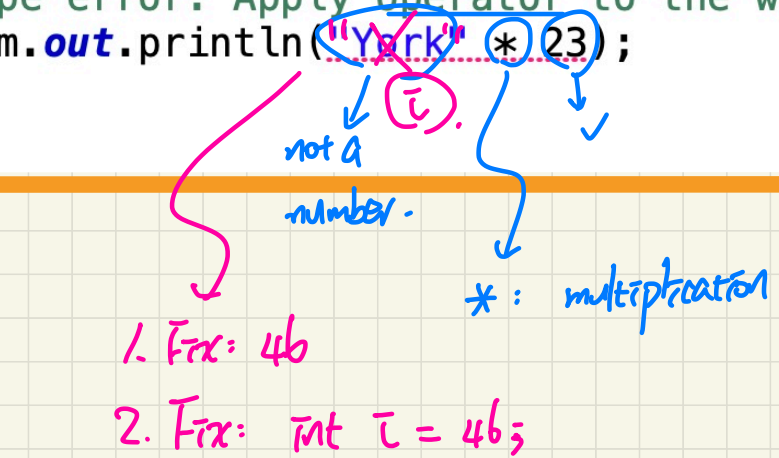
```
public class CompileTimeSyntaxError4 {
    public static void main(String[] args) {
        System.out.println("Hello");

        /* Error 3: extra ending curly bracket */
        }
    }
}
```

no opening { to match

Error at Compile Time: Type Errors (1)

CompileTimeTypeError1.java

```
public class CompileTimeTypeError1 {
    public static void main(String[] args) {
        /* Type error: Apply operator to the wrong values */
        System.out.println("York" * 23);
    }
}
```



Error at Compile Time: Type Errors (2)

CompileTimeTypeError2.java

```
public class CompileTimeTypeError2 {
    public static void main(String[] args) {
        /* Type error: Refer to undeclared variable */
        int i = 23;
        System.out.println(j / 3);
    }
}
```

int j = i * 2;

undeclared
⇒ unknown.

Error at Run Time: Exception

RunTimeException.java

```java
public class RunTimeException {
    public static void main(String[] args) {
        /* Runtime exception: code compiles but crashes at runtime */
        System.out.println(10 / 0);
    }
}
```

no compile-time error ⇒ runnable executable.

0 → division

math: undefined

prog: crash.

Error at Run Time: Logical Error

RunTimeLogicalError.java

```java
import java.util.Scanner;

public class RunTimeLogicalError {
    public static void main(String[] args) {
        /* Runtime logical error: code compiles, does not crash at runtime,
         * but does not behave as expected.
         */
        Scanner input = new Scanner(System.in);

        System.out.println("Enter the integer radius of a circle:");
        int radius = input.nextInt();

        System.out.println("Area of circle is: " + (2 * 3.14 * radius));
        input.close();
    }
}
```

logical error
wrong formula.

radius * radius * 3.14

1. Compiles
2. terminates without crashing
3. output is wrong.

Document Your Code

Single-Lined Comments: [Eclipse: `Ctrl + /`]

```
// This is Comment 1.
... // Some code
// This is Comment 2.
```

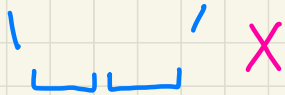
Multiple-Lined Comments: [Eclipse: `Ctrl + /`]

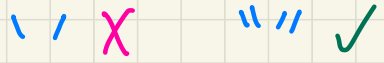
```
/* This is Line 1 of Comment 1.
 */
... // Some code
/* This is Line 1 of Comment 2.
 * This is Line 2 of Comment 2.
 * This is Line 3 of Comment 2.
 */
```

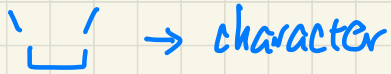
Lecture 1

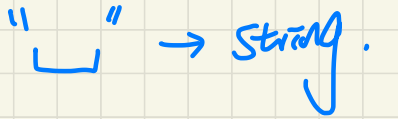
Part B

Elementary Programming - Literals, Operations

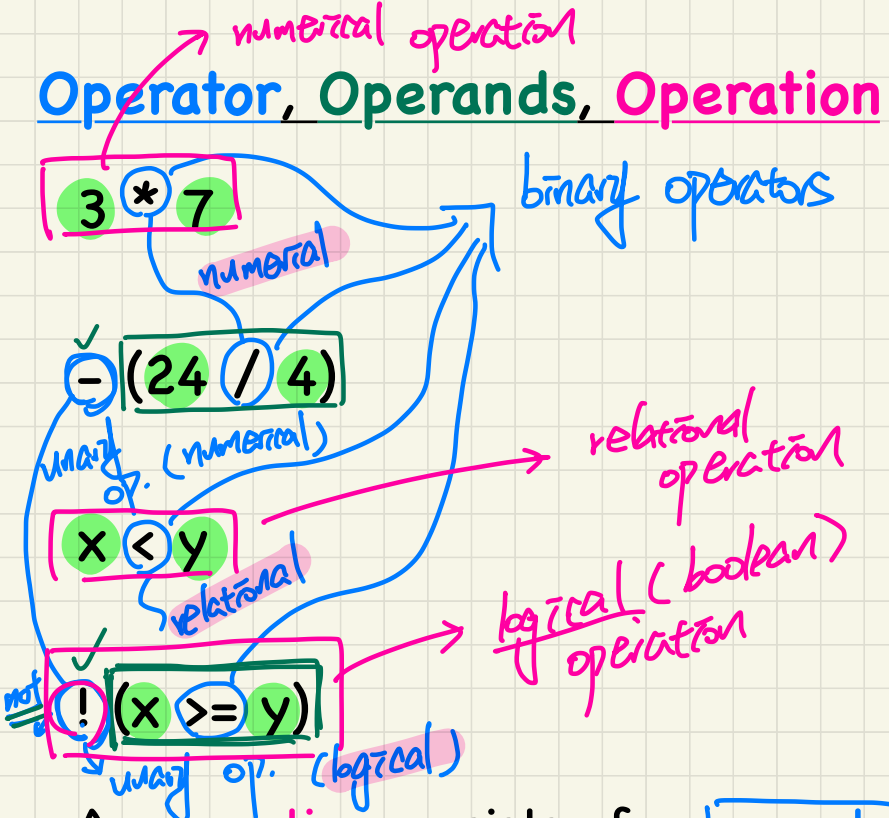








Operator, Operands, Operation



[>
<=
>=] ~ ! negation

[&& . and
|| . or] binary, logical.

- overloaded
e.g. -2 unary
e.g. 2-3 binary.

- An operation consists of an operator and one or more operands.
- An operator has one or more applicable operands. (unary vs. binary)
- An operation produces a value of certain type.

operation → op, operands

Division

Case 1 → both operands are integers

Given two integers x, y

23 4

x == y * (x / y) + (x % y)

5 3

23 / 4 → quotient → 5 with remainder 3.

23 % 4 → modulo remainder.

Case 2 → at least one operand is floating-point

23.0 / 4 → precise result

23 / 4.0

23.0 / 4.0

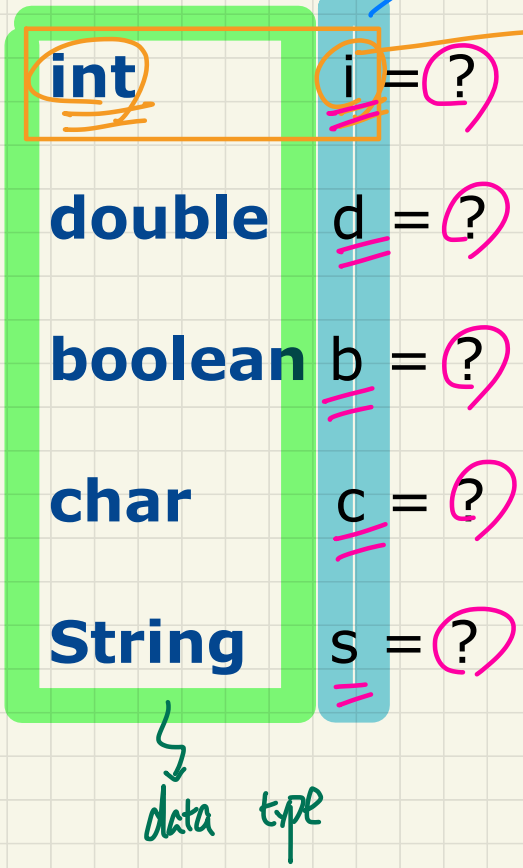
→ 5.75.

Lecture 1

Part C

Elementary Programming -
Data Types
Assignments, Constants vs. Variables

Data Type Declarations

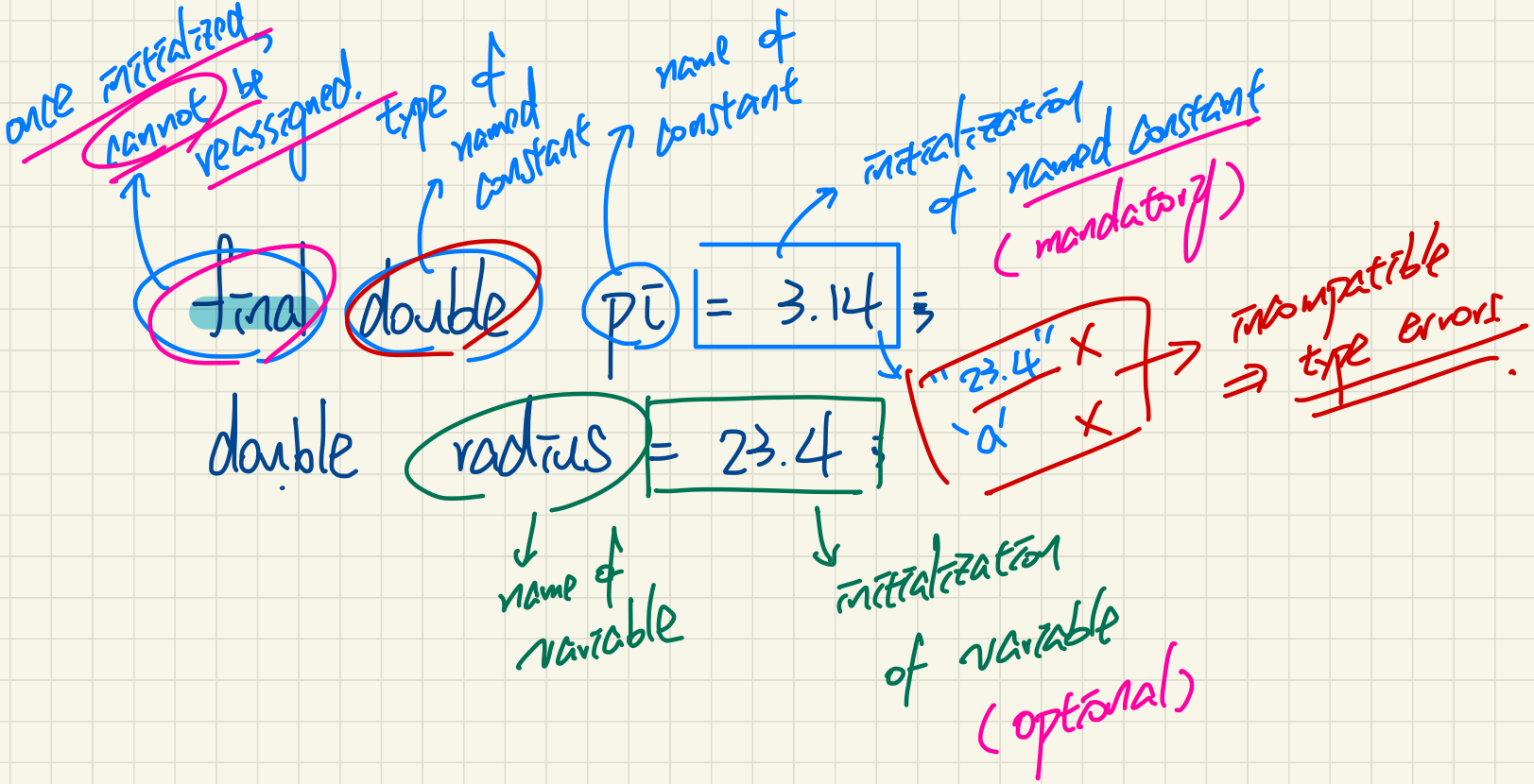


Consequence of declaring variable with name i of type int:

At runtime, only integer values can be stored in i.

i = "1022" ✗

once declared, cannot change the type of a variable.

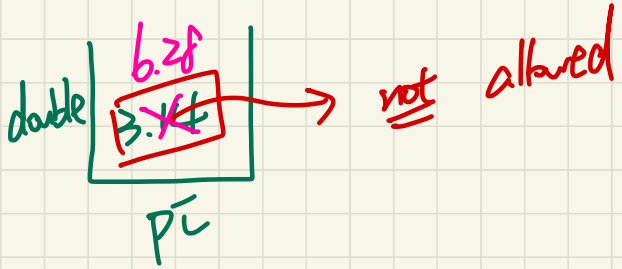

once initialized, cannot be reassigned.
type of named constant
name of constant
initialization of named constant (mandatory)
final
double
pi
= 3.14;
"23.4"
'a'
X
X
incompatible type errors.
double
radius
= 23.4;
name of variable
initialization of variable (optional)

Constant: Initialization vs. Re-Assignments

ConstantCannotBeReassigned.java

```
public class ConstantCannotBeReassigned {
    public static void main(String[] args) {
        /* A constant can only be initialized once. */
        final double pi = 3.14;
        /* Reassignment of a constant is illegal. */
        pi = 6.28;
    }
}
```

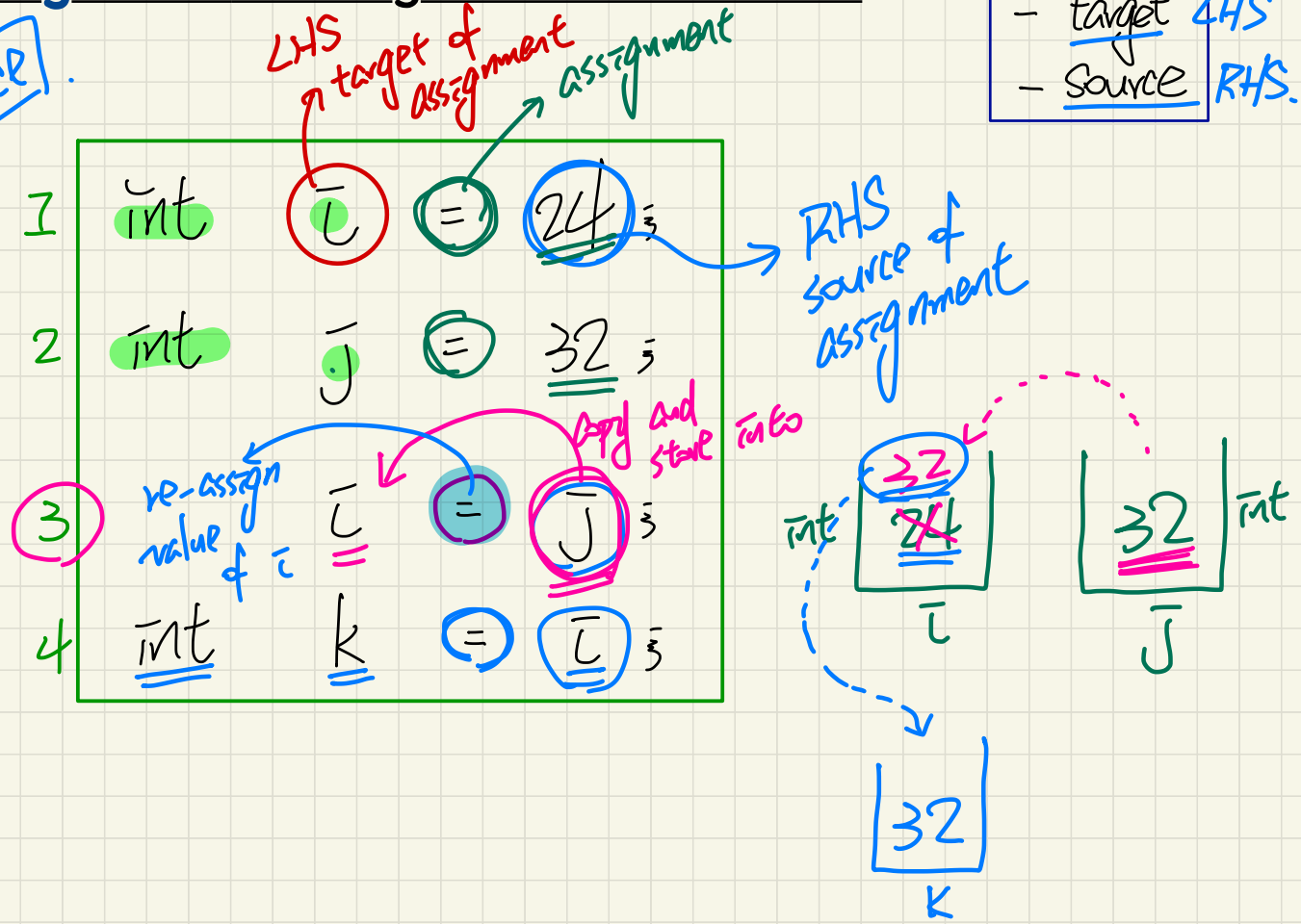
re-assignment

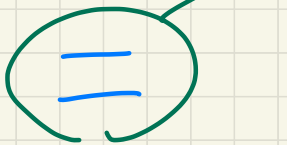


Assignment: Change of Stored Value

- type
- target LHS
- source RHS.

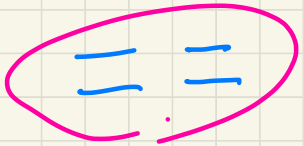
trace





assignment operator -

assignment



equal (value comparison)

relational operator

↳ T or F.

Lecture 1

Part D

Elementary Programming - Variable Re-Assignments Expressions, Type Correctness

Variable: Initialization vs. Re-Assignments

VariableCanBeReassigned.java

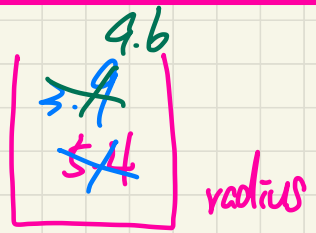
```
public class VariableCanBeReassigned {
    public static void main(String[] args) {
        /* A variable can be initialized. */
        double radius = 5.4;
        System.out.println("Radius is: " + radius);

        /* A variable may be re-assigned for as many times as necessary */
        radius = 3.9;
        System.out.println("Radius is: " + radius);
        System.out.println("Radius is: " + radius);

        radius = 9.6;
        System.out.println("Radius is: " + radius);
    }
}
```

5.4

3.9

3.9

9.6



Combining Constants and Variables

e.g., Print statements involving literals or named constants only:

```
final double PI = 3.14; /* a named double constant */
System.out.println("Pi is " + PI); /* str. lit. and num. const. */
System.out.println("Pi is " + PI);
```

"3.14"

PI 가 3.14

e.g., Print statements involving variables:

```
String msg = "Counter value is "; /* a string variable */
int counter = 1; /* an integer variable */
System.out.println(msg + counter);
System.out.println(msg + counter);
counter = 2; /* re-assignment changes variable's stored value */
System.out.println(msg + counter);
```

Counter value 가 1

1 → 2

counter

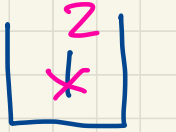
Counter value 가 2

Common Mistake: Declaring the Same Variable More Than Once

```
int counter = 1;
int counter = 2;
```

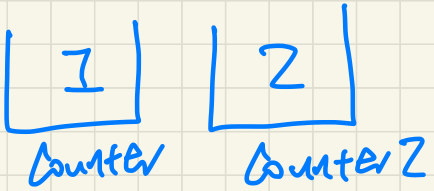
Fix 1: Only Keep the 1st Declaration

```
int counter = 1;
counter = 2;
```

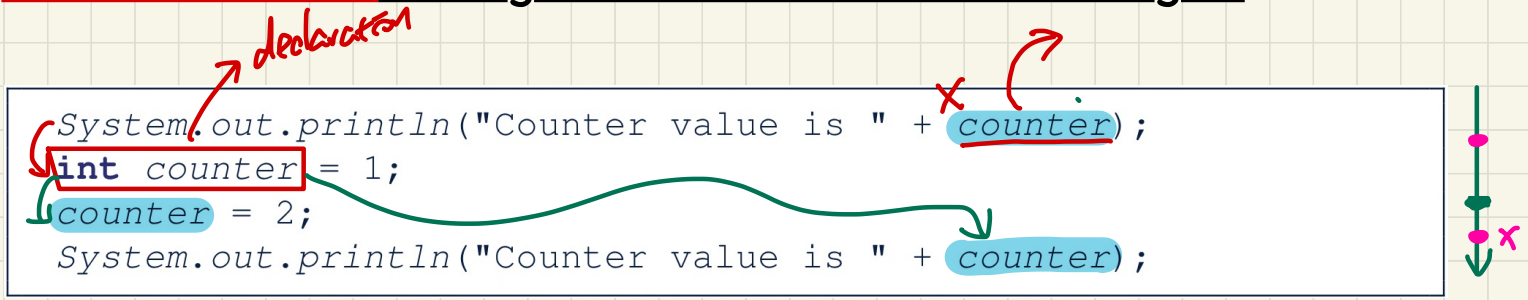


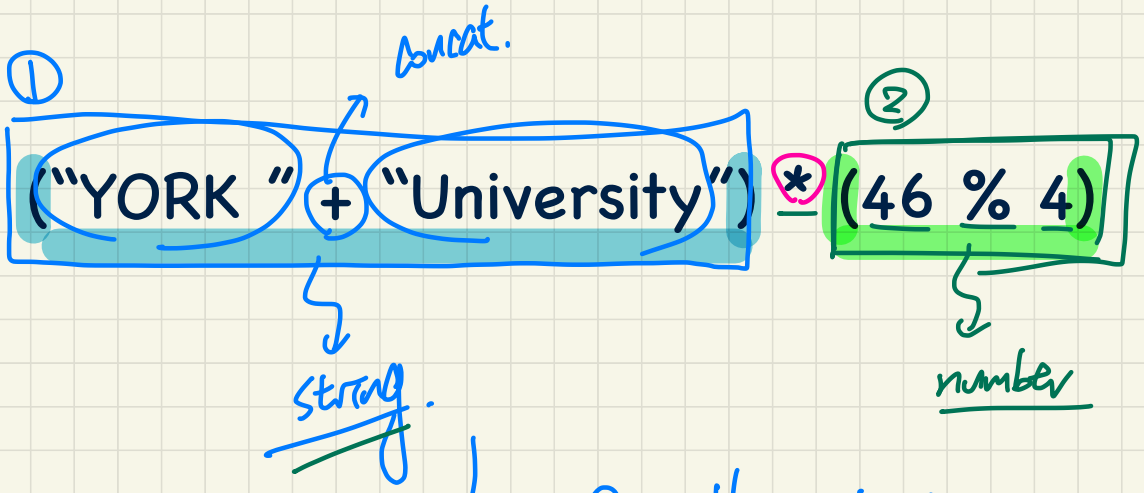
Fix 2: Declare a New Variable

```
int counter = 1;
int counter2 = 2;
```



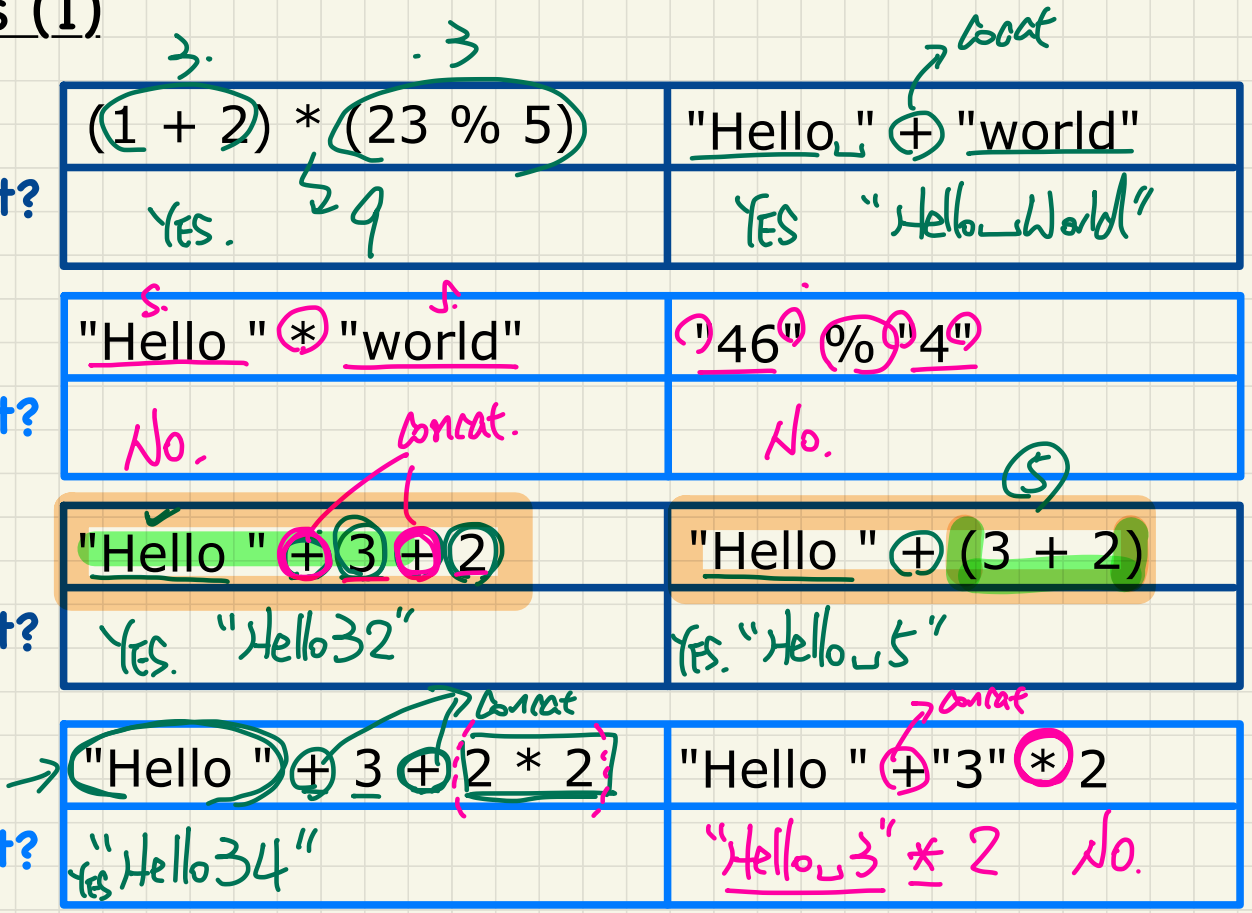
Common Mistake: Using a Variable Before Declaring It



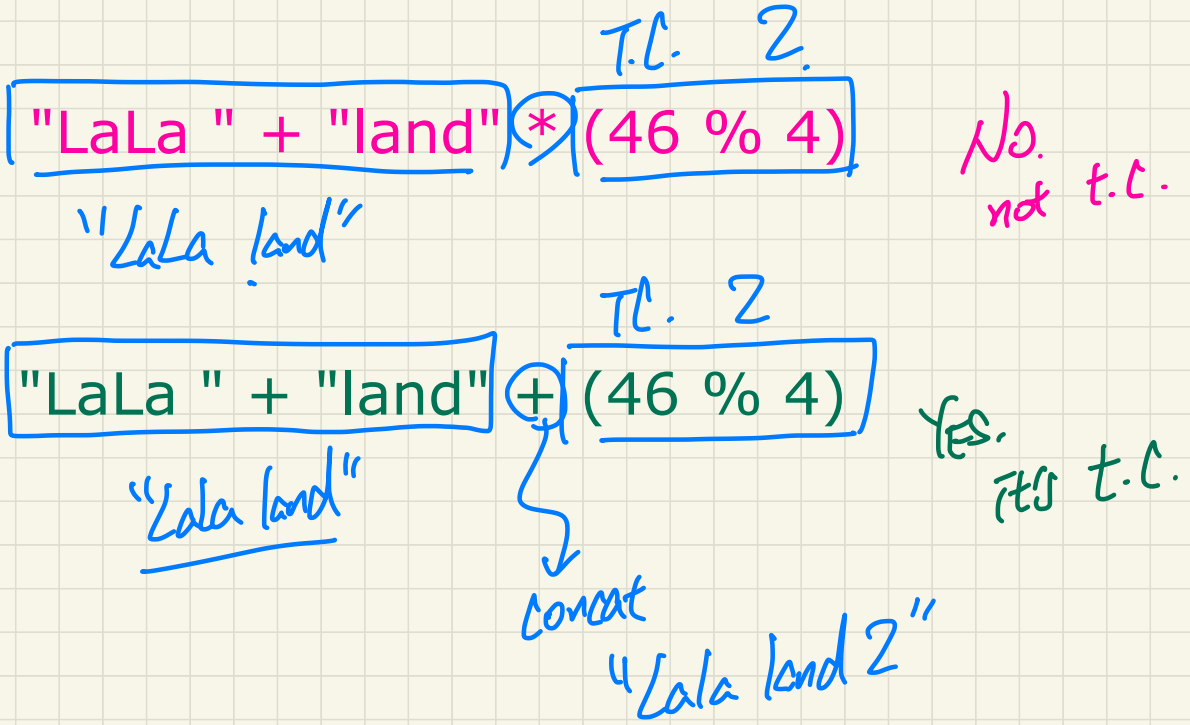


↳ Overall not type correct
even though sub-expressions
① ② are type-correct.

Expressions (1)



Expressions (2)



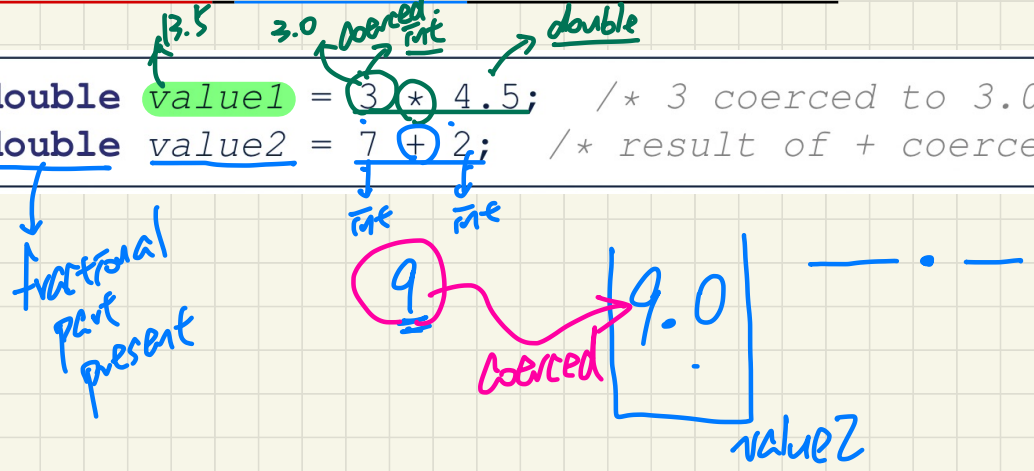
Lecture 1

Part E

Elementary Programming - Coercion vs. Casting

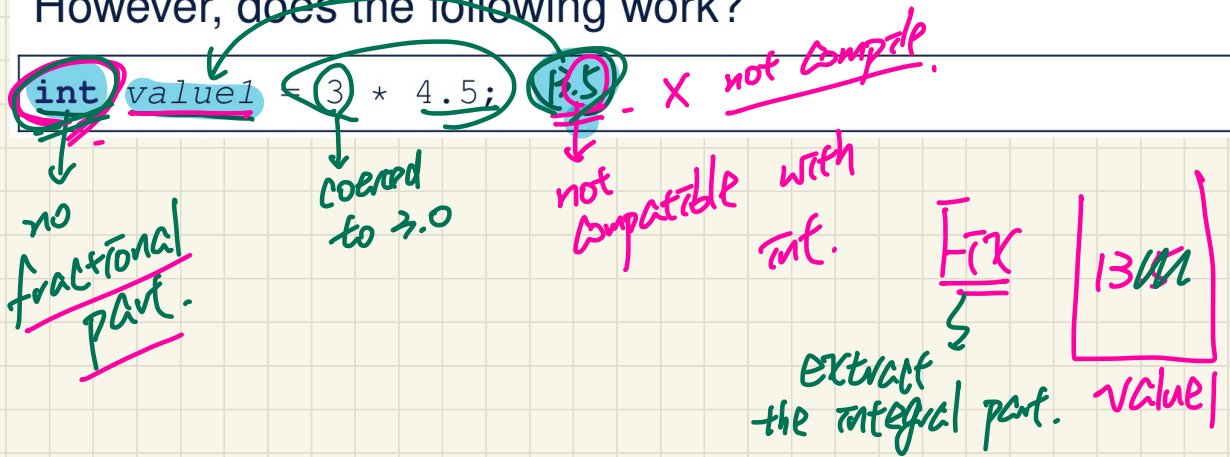
Automatic Coercion: int to double

```
double value1 = 3 * 4.5;   /* 3 coerced to 3.0 */
double value2 = 7 + 2;     /* result of + coerced to 9.0 */
```



However, does the following work?

```
int value1 = 3 * 4.5;
```



Manual Casting: double to int

Case 1: double to int

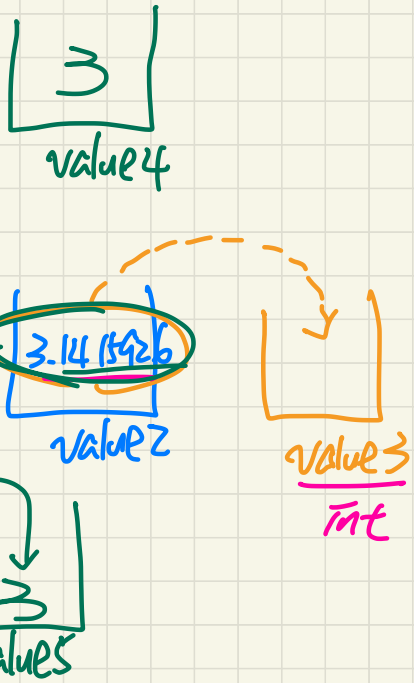
① `int value1 = 3.1415926;` ✗

② `int value4 = (int) 3.1415926;` ✓

③ `double value2 = 3.1415926;`

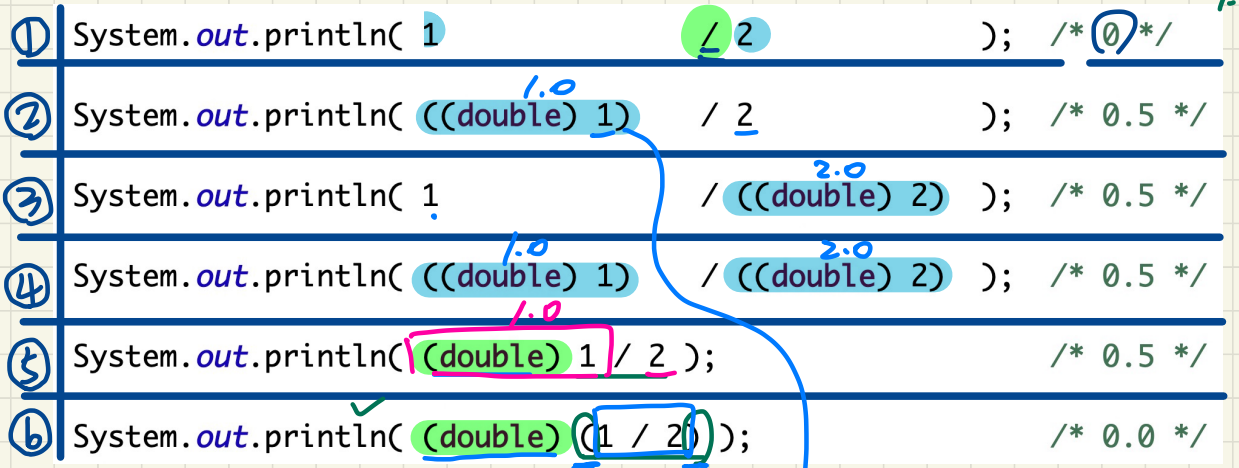
④ `int value3 = value2;` ✗

⑤ `int value5 = (int) value2;` ✓



Manual Casting: int to double

Case 1: int to double



(double) 1 / 2) equivalent ((double) 1) / 2

1 % 2 → 1

0.5

double v1 = 1; — coerced to 1.0

print(v1 / 2) → 0.5

int v2 = 1; — no coercion

print(v2/2) → 0

cast 0.0

precedence

3 * (2 + 3) — higher

(double) 1 → 1.0

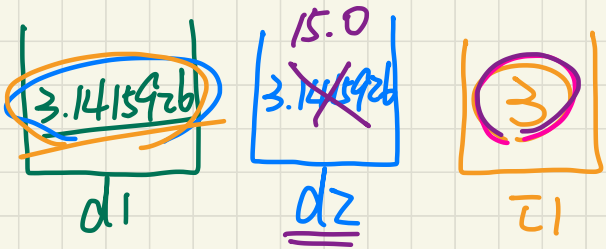
Exercise: Tracing Program

Consider the following Java code:

```
1  double d1 = 3.1415926;
2  System.out.println("d1 is " + d1);
3  double d2 = d1;
4  System.out.println("d2 is " + d2);
5  int i1 = (int) d1;
6  System.out.println("i1 is " + i1);
7  d2 = i1 * 5;
8  System.out.println("d2 is " + d2);
```

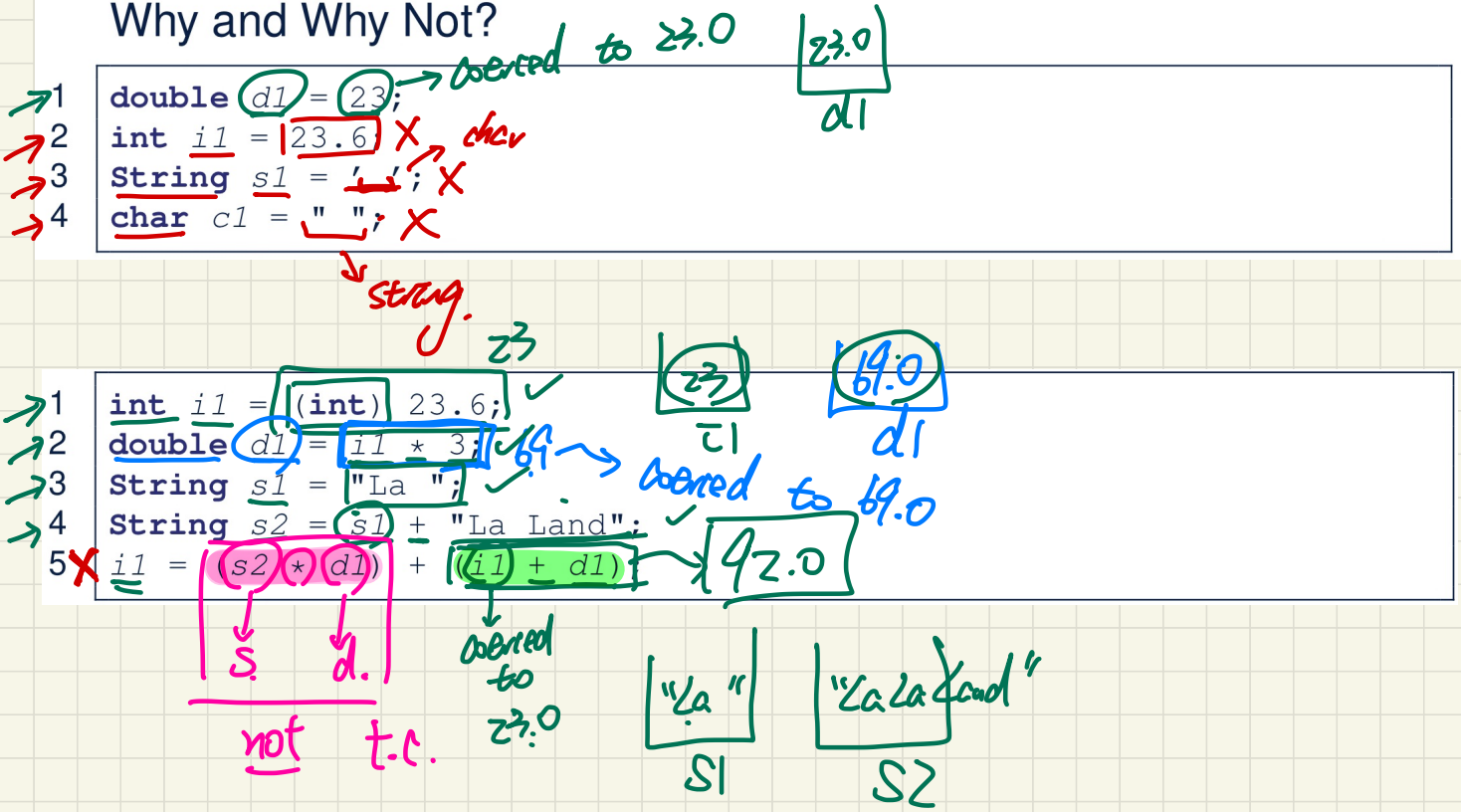
Write the **exact** output to the console.

```
d1 is 3.1415926
d2 is 3.1415926
i1 is 3
d2 is 15.0
```



Exercise: Type Correctness

Consider the following Java code, is each line type-correct? Why and Why Not?

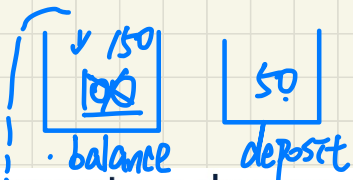


Lecture 1

Part F

Elementary Programming -
Augmented Assignments
Escape Sequences

Augmented Assignments



- You very often want to increment or decrement the value of a variable by some amount.

```
balance += balance + deposit;
balance = balance - withdraw;
```

150 50 150

syntactic sugar.

- Java supports special operators for these:

```
balance += deposit; // balance = balance + deposit
balance -= withdraw;
```

balance *= deposit
balance /= deposit

- Java supports operators for incrementing or decrementing by 1:

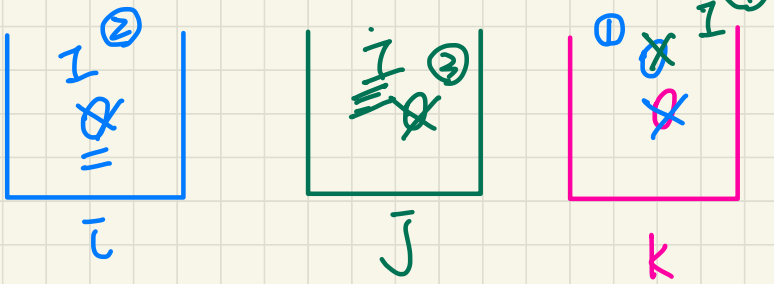
```
i ++; j --;
```

① ② i = i + 1 ③ i += 1

i ** X

Exercise: Preceeding vs Following ++

```
int i = 0; int j = 0; int k = 0;
k = i ++; /* k is assigned to i's old value */
k = ++ j; /* k is assigned to j's new value */
```



k = i ++;

↳ ① use i's value for assignment to k

→ ② perform ++

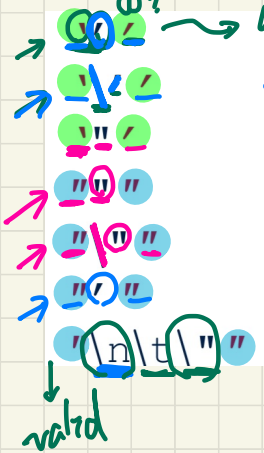
k = ++ j;

→ ③ perform ++ assgn. to k.

→ ④ use j's (new) value for

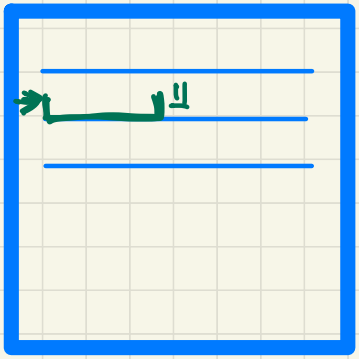
Escape Sequence

parse



ambiguity
- ① 2nd ', denotes end of char literal
- ② 2nd ', denotes "content" of char literal

[INVALID; need to escape ']
[VALID]
[VALID; no need to escape "]
[INVALID; need to escape "]
[VALID]
[VALID; no need to escape ']
[VALID]



Lecture 1

Part G

Elementary Programming - Sources for Variable Assignments

Console Application: With User Inputs vs Without

Without User Input

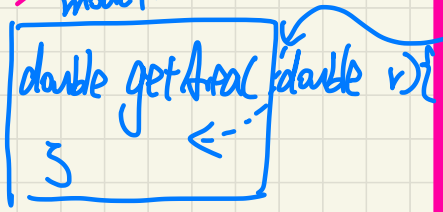
```
public class ComputeArea {
  public static void main(String[] args) {
    double radius; /* Declare radius */
    double area; /* Declare area */
    /* Assign a radius */
    radius = 20; /* assign value to radius */
    /* Compute area */
    area = radius * radius * 3.14159;
    /* Display results */
    System.out.print("The area of circle with radius ");
    System.out.println(radius + " is " + area);
  }
}
```

console apps

With User Input

```
import java.util.Scanner;
public class ComputeAreaWithConsoleInput {
  public static void main(String[] args) {
    /* Create a Scanner object */
    Scanner input = new Scanner(System.in);
    /* Prompt the user to enter a radius */
    System.out.print("Enter a number for radius: ");
    double radius = input.nextDouble();
    /* Compute area */
    final double PI = 3.14169; /* a named constant for π */
    double area = PI * radius * radius; /* area = πr² */
    /* Display result */
    System.out.println(
      "Area for circle of radius " + radius + " is " + area);
  }
}
```

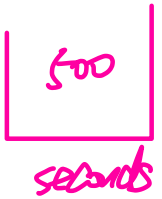
model.



Example: Convert Seconds to Minutes

```
import java.util.Scanner;
public class DisplayTime {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    /* Prompt the user for input */
    System.out.print("Enter an integer for seconds: ");
    int seconds = input.nextInt();
    int minutes = seconds / 60; /* minutes */
    int remainingSeconds = seconds % 60; /* seconds */
    System.out.print(seconds + " seconds is ");
    System.out.print(" minutes and ");
    System.out.println(remainingSeconds + " seconds");
  }
}
```

Test: 500 seconds



int: 500 int quotient 6

20

Exercise: Modify the program so that it will display hours if necessary.

e.g., 7945 seconds -> 2 hours, ~~2~~ 12 minutes, ~~8~~ 25 seconds

Where Can An Assignment Source (RHS) Come From?

In `tar = src`, the *assignment source* `src` may come from:

- A literal

```
int i = 23;
```

- A variable

```
int i = 23;
int j = i;
```

- An expression involving literals and variables

```
int i = 23;
int j = i * 2;
```

`(i / j) * (i % j)` — int

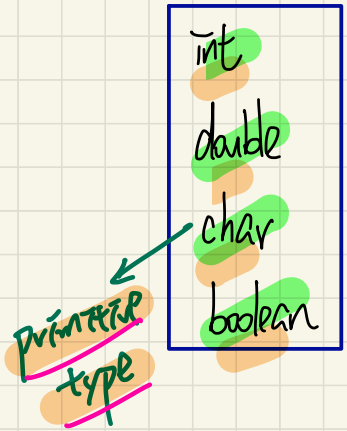
type of expression must match the declared type of assignment target

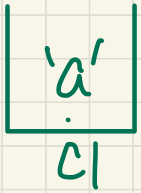
- An input from the user

```
Scanner input = new Scanner(System.in);
int i = input.nextInt();
int j = i * 2;
```

int

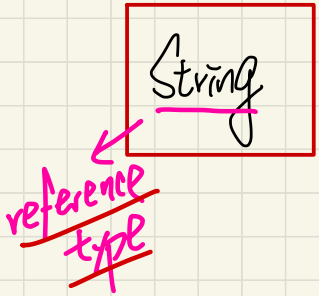
Comparison of Values





use ==

e.g., char c1 = 'a';
println (c1 == 'b');



use equals

s1 == "quit" X
- compile
- won't crash
- won't work → logical error.

e.g., String s1 = input.nextLine();
println (s1.equals("quit"));

Printing to Console

```
String s1 = "A";
String s2 = "B";
```

print(s1)
println(s2)

```
print (s1);
print (s2);
```

AB

AB.

```
println (s1);
println (s2);
```

A
B

```
print (s1 + "\n");
println (s2);
```

A
B