Java Review – Variables

CPSC 1181 – O.O.

Jeremy Hilliker

Summer 2017

Langara.

THE COLLEGE OF HIGHER LEARNING.

Types

- A Type is:
  - "A set of values and the operations on those values"

- EG:
  - Integer types: byte, char*, short, int, long
  - Floating points: float, double
  - Boolean: bool (logical operators, assignment, "==")
  - Objects: String, OutputStream, InputStream, Scanner, …
    - String: "+", ".", "=", "=="

Primitive Types

Identifiers

- An “identifies” is the **name** of a thing.
- There are formal rules for them.
  - Can be made up of letters, digits, and the underscore (_) character
  - Cannot start with a digit
  - Cannot use other symbols such as ? or %
  - Spaces are not permitted inside identifiers
  - You cannot use reserved words
  - They are case sensitive

- By convention, **variable names** start with a lowercase letter
- By convention, **class names** start with an uppercase letter

```
String greeting = "Hello, World!";
PrintStream printer = System.out;
int luckyNumber = 13;
```

Assignment Operator

- " = "
- Sets / changes the value of a variable
  - int x = 1; Expression
  - x = $\boxed{\text{x + x;}}$
- Expression: a thing that **has a value**

Casting

- Used to convert one type to another
  - They must be compatible / convertible
  - **`(`*`type`*`)` *`expression`***

- eg:
  - (int) (4.0 / 2.0)
  - String s = (String) x;

final

- A keyword to indicate that a variable does not change

```
final int foreverFive = 5;
```

static

- A keyword to indicate that there should only be one of these things

```
public static int sharedValue = 42;
```

Constants

- Are both **static** and **final**
- Written in all UPPER_CASE

```
public final static float MAX_GRADE = 100.0;
```

Static Method

- Saying that this method is not associated with an instance of the class
- ie: it may only locally access other static methods or variables
  - “shared” methods and variables
- Non-static methods may only be called on objects.

Strings

- A sequence of character
- Objects of the String class (type)
- Eg
  - null // no string
  - “” // empty
  - “hello”
  - String msg = “boo”;
  - int n = s.length()

Concatenation

- The “ + “ operator

- Eg
  - “Dave“ + “ is not a cool dude.”
  - str1 + str2
  - str1 + 7

Converting

```
String s = “7”
int n = Integer.parseInt(s);

String str = “” + 7
Str = Integer.toString(n);
```

Substrings

String greeting = "Hello, World!";

String sub = greeting.substring(0, 5); // sub is "Hello”

String s = greeting.substring(7, 12); // s is “World"

- Supply start and “past the end” position

- Or:

String s1 = greeting.substring(7); // s1 is “World!"

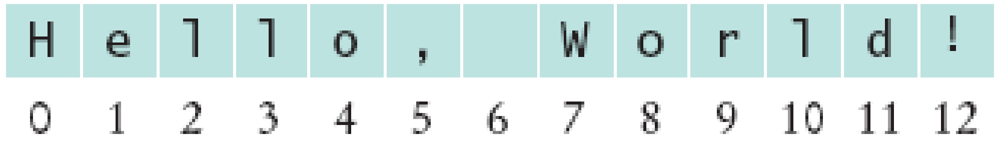


Figure 3 String Positions

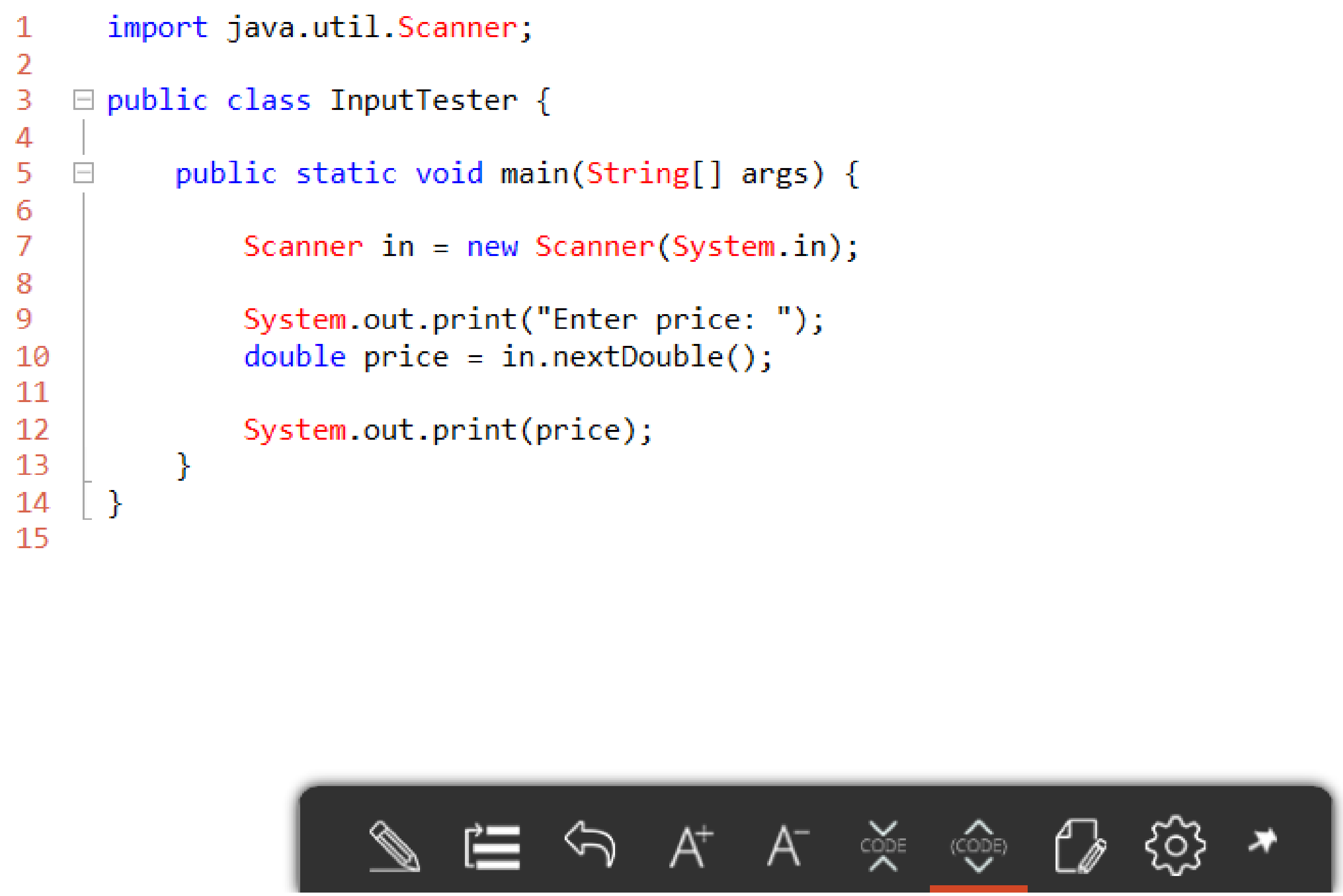
$$\underbrace{\underbrace{\texttt{(-b + }\underbrace{\texttt{Math.sqrt(}\underbrace{\underbrace{\texttt{b * b}}_{b^2}\texttt{ - }\underbrace{\texttt{4 * a * c}}_{4ac}}_{b^2-4ac}\texttt{)}}_{\sqrt{b^2-4ac}}\texttt{)}}_{-b+\sqrt{b^2-4ac}}\texttt{ / (}\underbrace{\texttt{2 * a}}_{2a}\texttt{)}}_{\frac{-b+\sqrt{b^2-4ac}}{2a}}$$

Reading Input

- `System.in` has minimal set of features–it can only read one byte at a time
- In Java 5.0, **`Scanner`** class was added to read keyboard input in a convenient manner

```
Scanner in = new Scanner(System.in);
System.out.print("Enter quantity: ");
int quantity = in.nextInt();
```

- **`nextDouble`** reads a double
- **`nextLine`** reads a line (until user hits Enter)
- **`nextWord`** reads a word (until any white space)



```java
import java.util.Scanner;

public class InputTester {

    public static void main(String[] args) {

        Scanner in = new Scanner(System.in);

        System.out.print("Enter price: ");
        double price = in.nextDouble();

        System.out.print(price);
    }
}

```

Java	Math Notation	Description
>	$>$	Greater than
>=	$\geq$	Greater than or equal
<	$<$	Less than
<=	$\leq$	Less than or equal
==	$=$	Equal
!=	$\neq$	Not equal

(Identical) Comparisons

-
  ```
  a = 5; // Assign 5 to a
  if (a == 5) . . . // Test whether a equals 5
  ```

```
double r = Math.sqrt(2);
double d = r * r -2;
if (d == 0)
   System.out.println("sqrt(2)squared minus 2 is 0");
else
   System.out.println("sqrt(2)squared minus 2 is not 0 but " + d);
```

```
final double EPSILON = 1E-14;
if (Math.abs(x - y) <= EPSILON)
   // x is approximately equal to y
```

Comparing Strings

```
if (input == "Y") // WRONG!!!
```

```
if (input.equals("Y"))
```

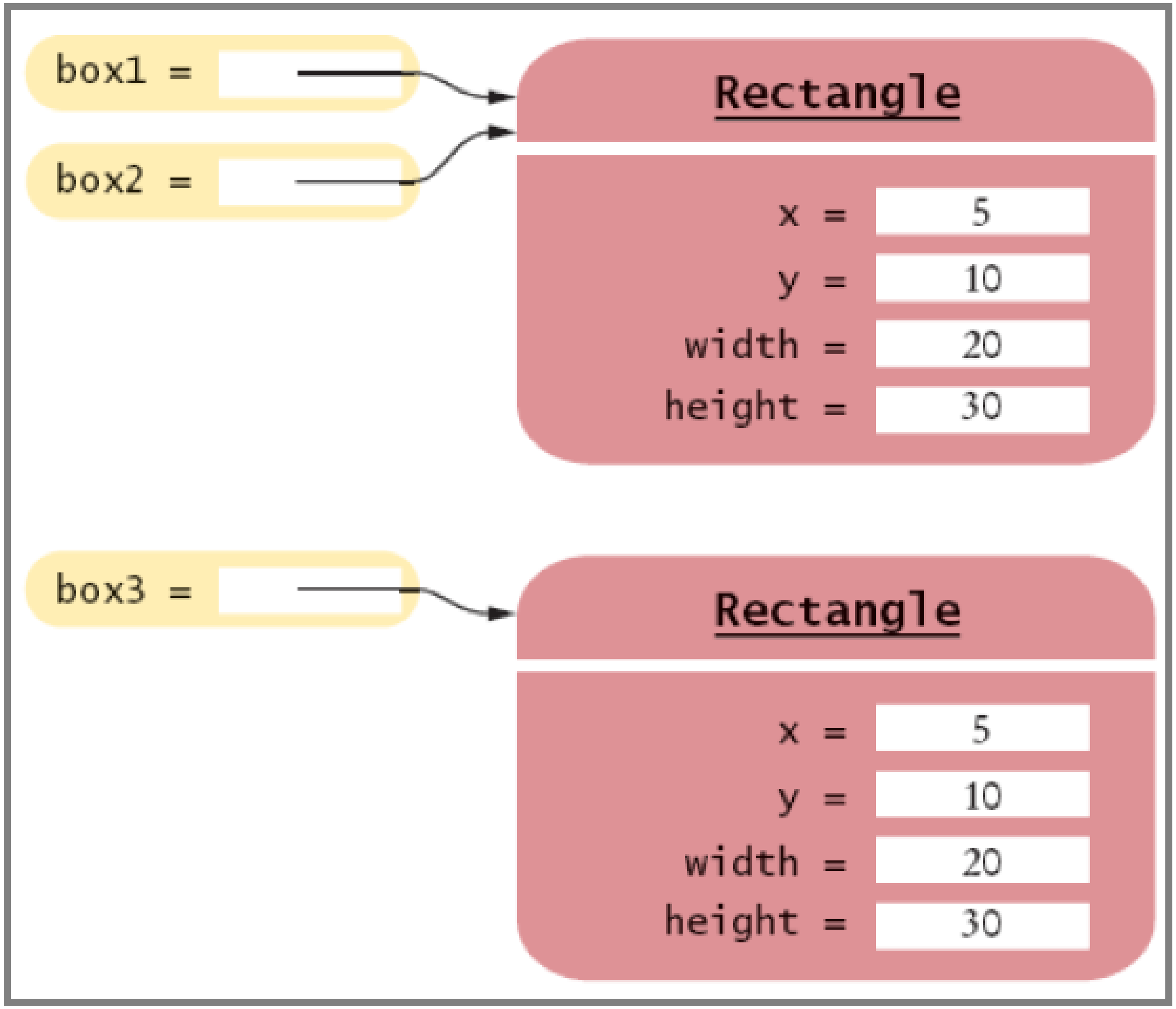
```
if (input.equalsIgnoreCase("Y"))
```

- s.compareTo(t) < 0 means
  s comes before t in the dictionary
- "car" comes before "cargo"
- All uppercase letters come before lowercase:
  "Hello" comes before "car"

- `==` tests for identity, `equals` for identical content

```
Rectangle box1 = new Rectangle(5, 10, 20, 30);
Rectangle box2 = box1;
Rectangle box3 = new Rectangle(5, 10, 20, 30);
```

- `box1 != box3` Is this right?
  but `box1.equals(box3)`
- `box1 == box2` Is this right?
- Note: `equals` must be defined for the class


box1 =
box2 =
Rectangle
x = 5
y = 10
width = 20
height = 30
box3 =
Rectangle
x = 5
y = 10
width = 20
height = 30

- `null` reference refers to no object
- Use **`==`, not `equals`, to test for `null`**
- **`null` is not** the same as the empty string `""`

- Sentinel value: Can be used for indicating the end of a data set
- `0` or `-1` make poor sentinels; better use `Q`

```
System.out.print("Enter value, Q to quit: ");
String input = in.next();
if (input.equalsIgnoreCase("Q"))
   We are done
else
{
   double x = Double.parseDouble(input);
   . . .
}
```