CS 171: Introduction to Computer Science II

Department of Mathematics and Computer Science

Li Xiong

Roadmap

- Lab session
- Pretest Postmortem
- Java Review
  - Types, variables, assignments, expressions
  - Control flow statements
  - Methods
  - Arrays
  - OO and Inheritance

Lab Session

- Option 1: Monday 10-11am
- **Option 2: Monday 5-6pm (selected)**
- Option 3: Friday 10-11am

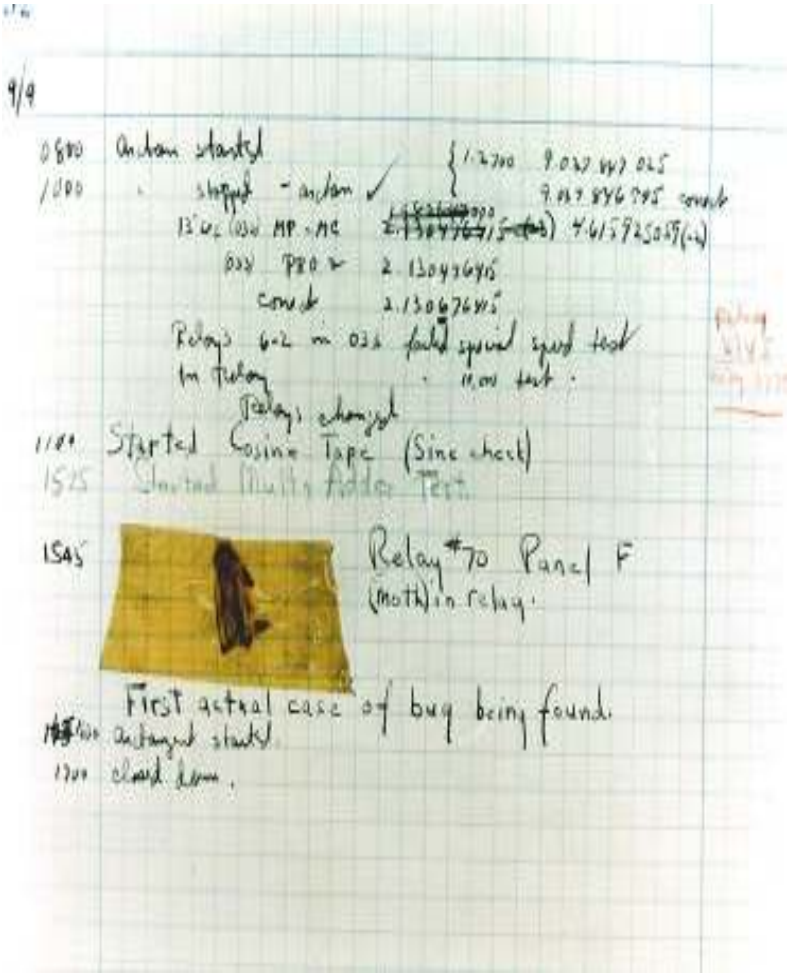
- First lab session: Eclipse and debugging lab

Debugging

- Debugging - finding and correcting errors
- Approaches
  - Hand-trace the program
  - Insert print statements to show the values of the variables or the execution flow

```
System.out.println("radius
= " + radius);
```

  - Use a debugger utility



Programming Errors

- Syntax Errors
  - Detected by the compiler
  - E.g. variable not initialized
- Runtime Errors
  - Causes the program to abort
  - E.g. array index out of bounds
- Logic Errors
  - Produces incorrect result

Roadmap

- Lab session
- Pretest Postmortem
- Java Review
  - Types, variables, assignments, expressions
  - Control flow statements
  - Methods
  - Arrays
  - OO and Inheritance

Pretest Postmortem

Question	Topics	#correct answers
1	Loops; post increment operator	15/31
2	Arithmetic operations - division; modulo	17/31
3	Object variables; null references	2/31
4	Integer variables	23/31
5	Object variables	23/31
6	Loops; arrays; problem solving	8/31
7a)	Inheritance	28/31
7b)	Inheritance; class constructor	11/31
7c)	Methods; overloading; polymorphism	1/31
7d)	Inheritance; problem solving	7/31

Roadmap

- Lab session
- Pretest Postmortem
- Java Review
  - Types, variables, assignments, expressions
  - Control flow statements
  - Methods
  - Arrays
  - OO and Inheritance

Data Types

- Primitive types
  - 6 numeric types
    - 4 integral types: `byte, short, int, long`
    - 2 floating point types: `float, double`
  - 1 character type: `char`
  - 1 boolean type: `boolean`
- Reference types
  - Class types, interface types, array types
  - Special `null` types

Variables

- A variable is a name for a location in memory used to hold a data value.
  - Type, name and contents
- Using a variable
  - Declaring a variable – type and name
    - Instructs the compiler to reserve a portion of main memory to hold a particular type of value referred by a particular name
  - Assign a value to a variable
  - Use a variable in an expression
    - A variable cannot be used if it is not declared or initialized
    - The left hand side of the assignment operator is always a variable and the right hand side is an expression

Using Variables

- Declaring a variable

```
int count;
```

- Assign a value to a variable

```
count = 0;
```

- Declaring and initializing in one step

```
int count = 0;
```

- Using a variable in an expression

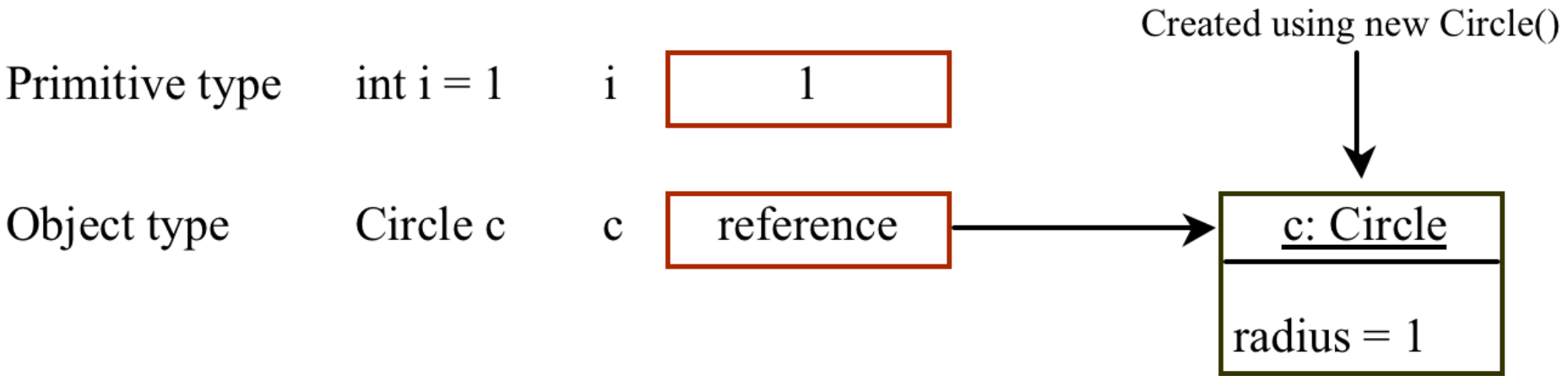
```
count += 1;
```

```
count ++;
```

Primitive data types vs. object data types

- Variables of primitive data types store the actual value
- Variables of object types store the reference to the object

```
int i = 1;
Circle c = new Circle();
```



The null Value

- If a data field of a reference type does not reference any object, the data field holds a special value: null

```
Circle c;
double r = c.getRadius();
```

Question

- A variable, int x stores: _____

A. A reference to an int
B. An integer value
C. The identifier, ”x”
D. Lots of goodies for every good Java-slave

Question

- A variable, BankAccount x stores: _____
  - A reference to an object of the BankAccount class
  - An object of the BankAccount class
  - The identifier, ”x”
  - Even more goodies than a mere int x

Question

- Which of the following will always correctly check whether an object variable obj contains a null reference? _____

  A) obj.equals(null);
  B) null == obj;
  C) obj = null;
  D) null.equals(obj);
  E) None of the above

Expressions

- An expression is a combination of one or more operators and operands that perform a calculation
  - Operands might be numbers, variables, or expressions
- Arithmetic expressions

```
int score = score - 10 * lateDays
```

- Boolean expressions

```
boolean isLate = submissionDate <= dueDate;
```

Numeric Operators

Name	Meaning	Example	Result
+	Addition	34 + 1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

- Division performs integer division when both operands are integers

Question

- If a and b are ints such that b != 0 then which of the following expressions is always equivalent to a%b? _____

A) a-(a/b)*b
B) (a/b)*b
C) a-a/b
D) (double)a/b - a/b

Shortcut Assignment Operators

Operator	*Example*	*Equivalent*
`+=`	`i += 8`	`i = i + 8`
`-=`	`f -= 8.0`	`f = f - 8.0`
`*=`	`i *= 8`	`i = i * 8`
`/=`	`i /= 8`	`i = i / 8`
`%=`	`i %= 8`	`i = i % 8`

Increment and Decrement Operators

Operator	Name	Description
++var	preincrement	The expression (++var) increments var by 1 and evaluates to the *new* value in var *after* the increment.
var++	postincrement	The expression (var++) evaluates to the *original* value in var and increments var by 1.
--var	predecrement	The expression (--var) decrements var by 1 and evaluates to the *new* value in var *after* the decrement.
var--	postdecrement	The expression (var--) evaluates to the *original* value in var and decrements var by 1.

Increment and Decrement Operators, cont.

```
int i = 10;
int newNum = 10 * i++;
```

Same effect as

```
int newNum = 10 * i;
i = i + 1;
```

```
int i = 10;
int newNum = 10 * (++i);
```

Same effect as

```
i = i + 1;
int newNum = 10 * i;
```

The Boolean Expressions

- A Boolean expression evaluates to a Boolean value
- Comparison operators: compare a pair of values (numbers, characters, boolean values)

```
boolean happy = grade > 90;
```

- Boolean operators: perform logic operations (boolean values)

```
boolean happy = (grade > 90)
&& (workhours < 2);
```

Comparison Operators

Operator	*Name*
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to

Comparing objects

- == compares references
  - Check whether an object variable contains a null reference
- equals() method compares contents
  - The default implementation of the equals method in the Object class:

```
public boolean equals(Object obj) {
  return (this == obj);
}
```

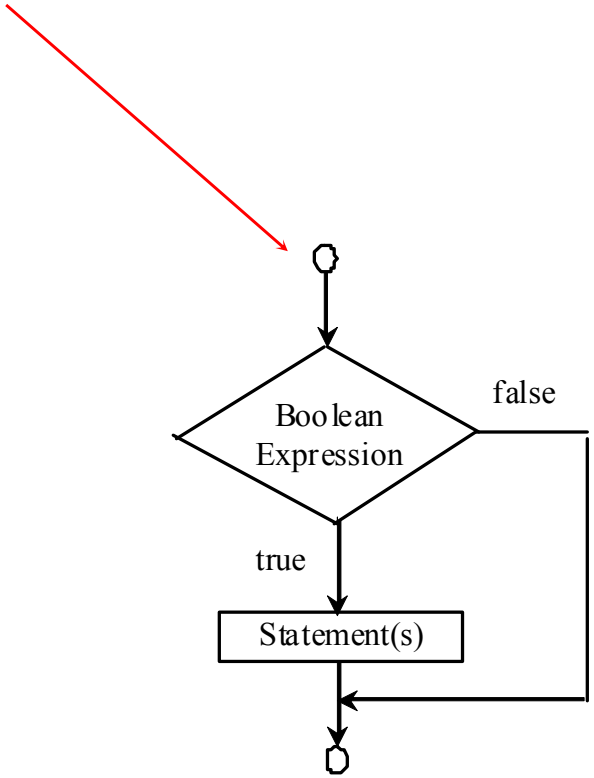
  - Java classes such as String override equals() method so that it compares the content of two objects.
  - It is a good idea to override equals() method for your own classes

Roadmap

- Lab session
- Pretest Postmortem
- Java Review
  - Types, variables, assignments, expressions
  - Control flow statements
  - Methods
  - Arrays
  - OO and Inheritance

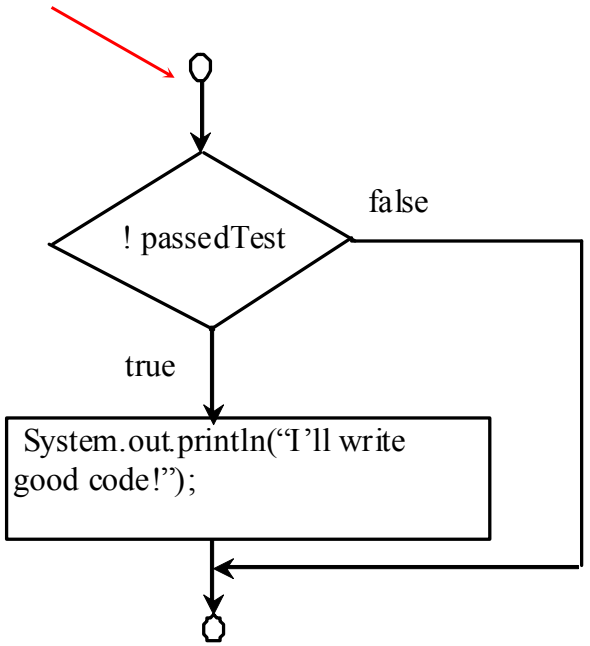
Simple if Statements

```
if (booleanExpression) {
  statement(s);
}
```



(A)

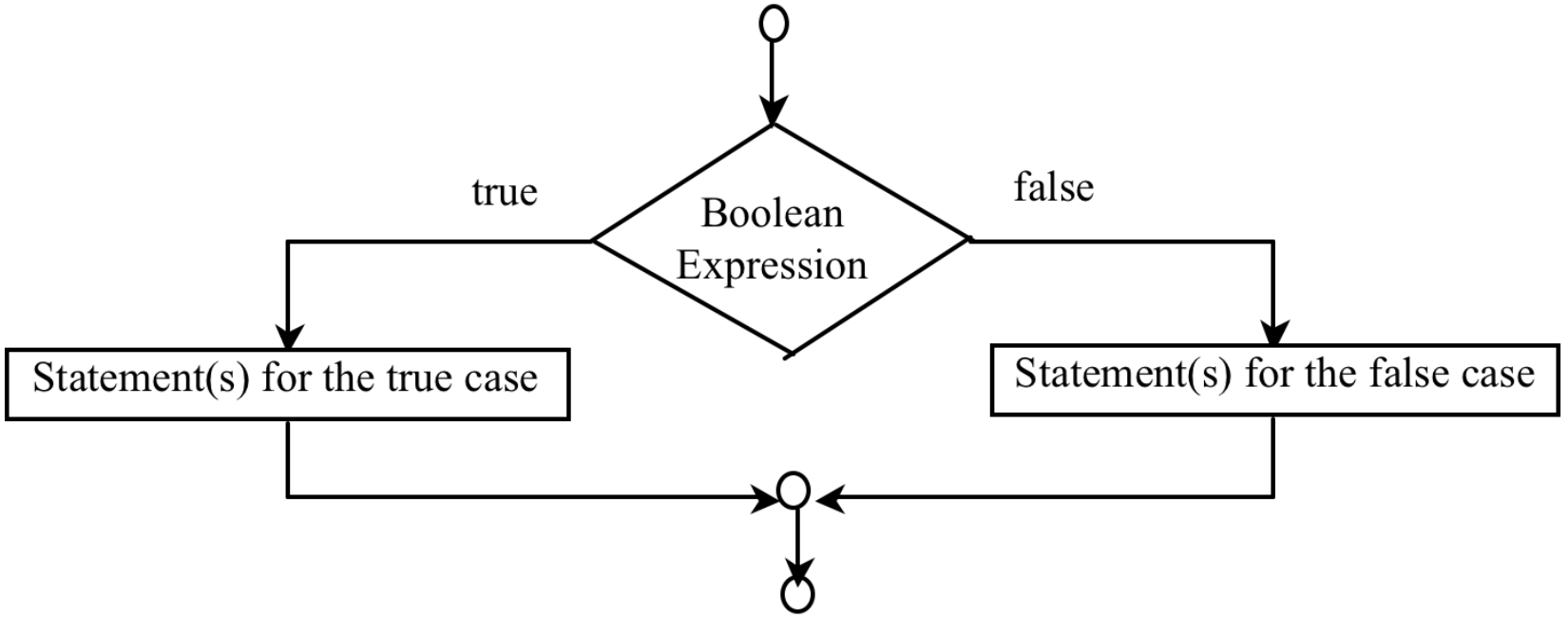
```
if ( !passedTest ) {
  System.out.println("I'll write
   good code!");
}
```



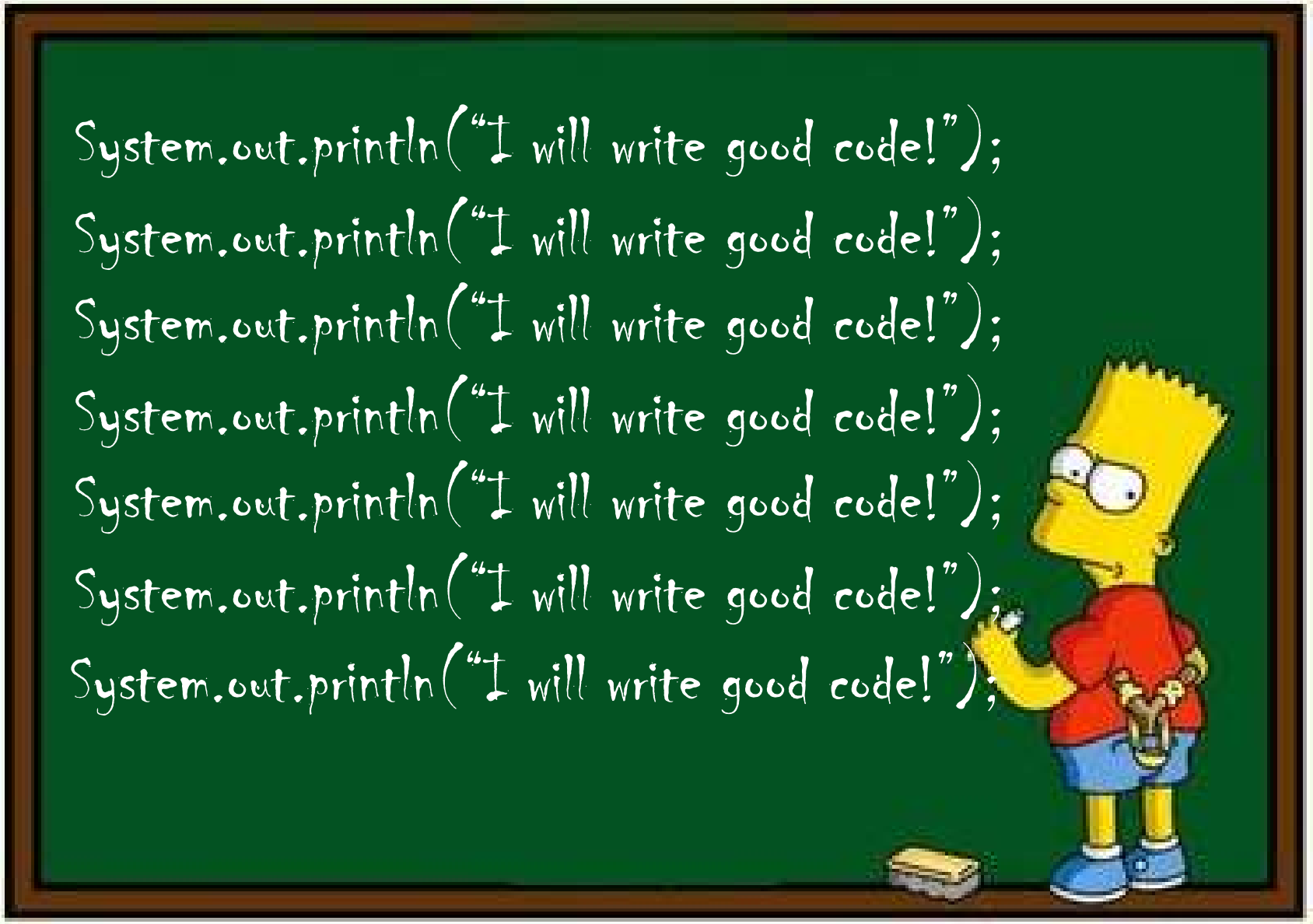
(B)

The if...else Statement

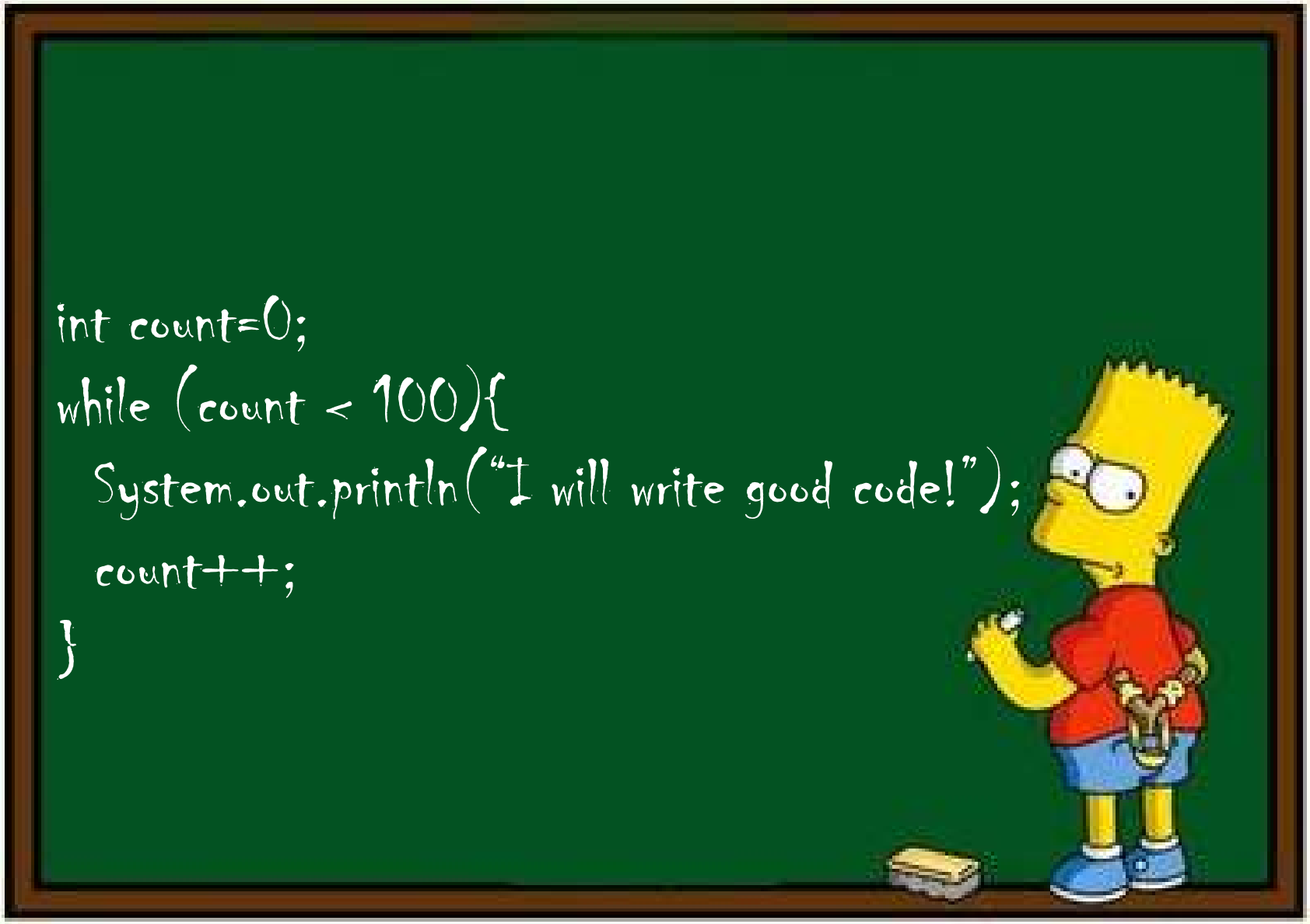
```
if (booleanExpression) {
  statement(s)-for-the-true-case;
}
else {
  statement(s)-for-the-false-case;
}
```



World Without Loops is Painful…



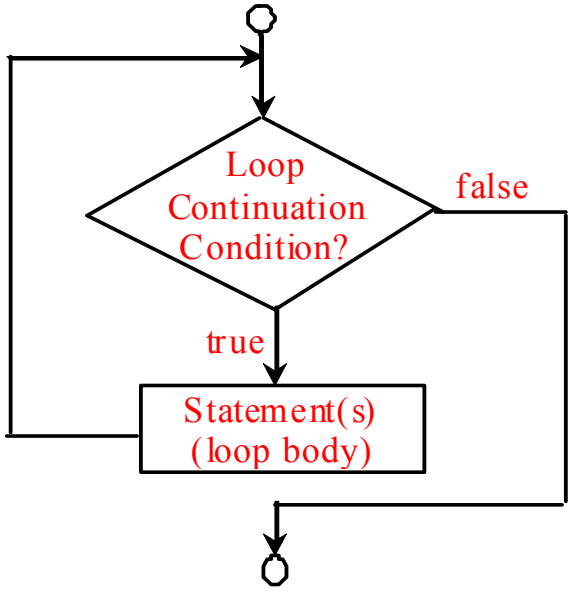
A Better Approach: **Loops**



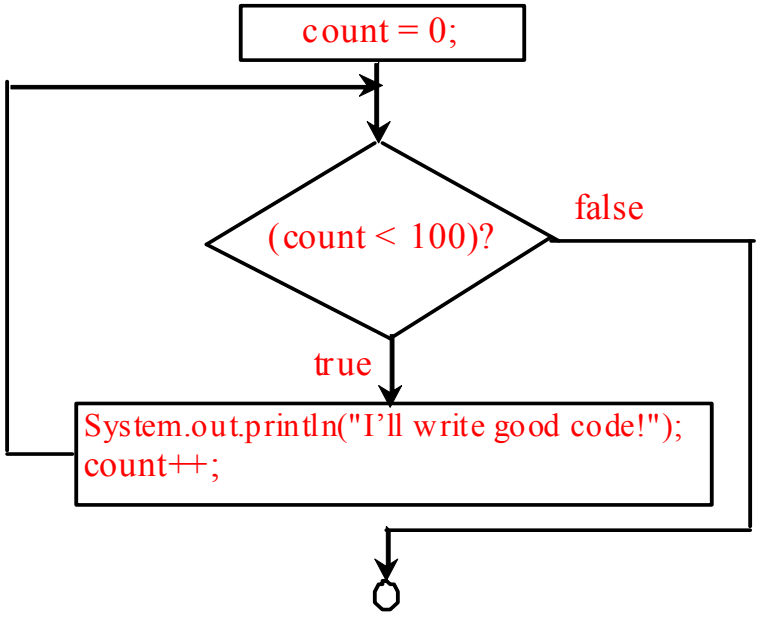
`while` Loop Flow Chart

```
while (loop-continuation-condition) {
  // loop-body;
  Statement(s);
}
```

```
int count = 0;
while (count < 100) {
  System.out.println("I'll write good code!");
  count++;
}
```

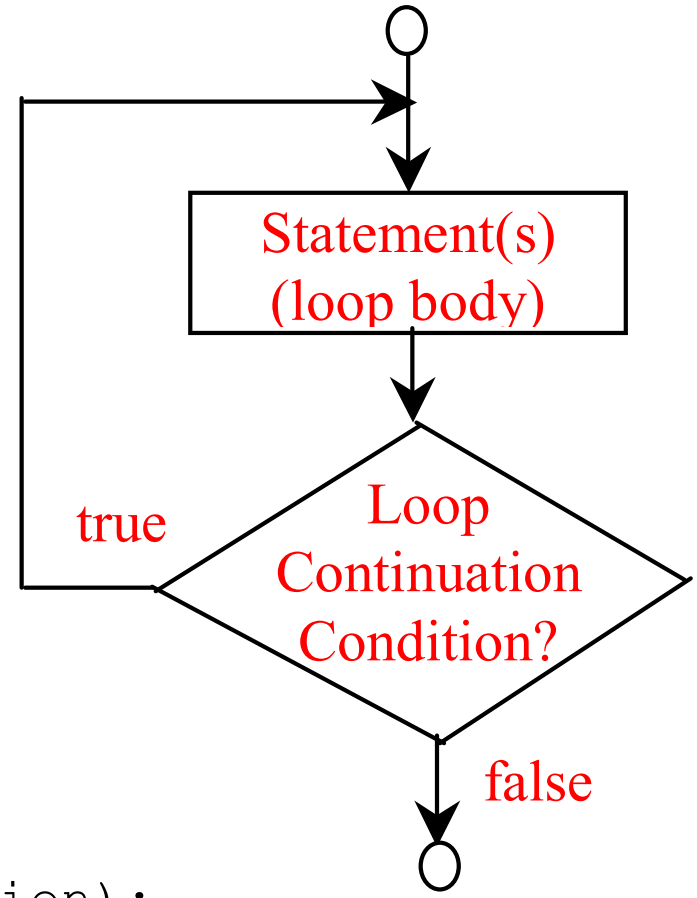


(A)



(B)

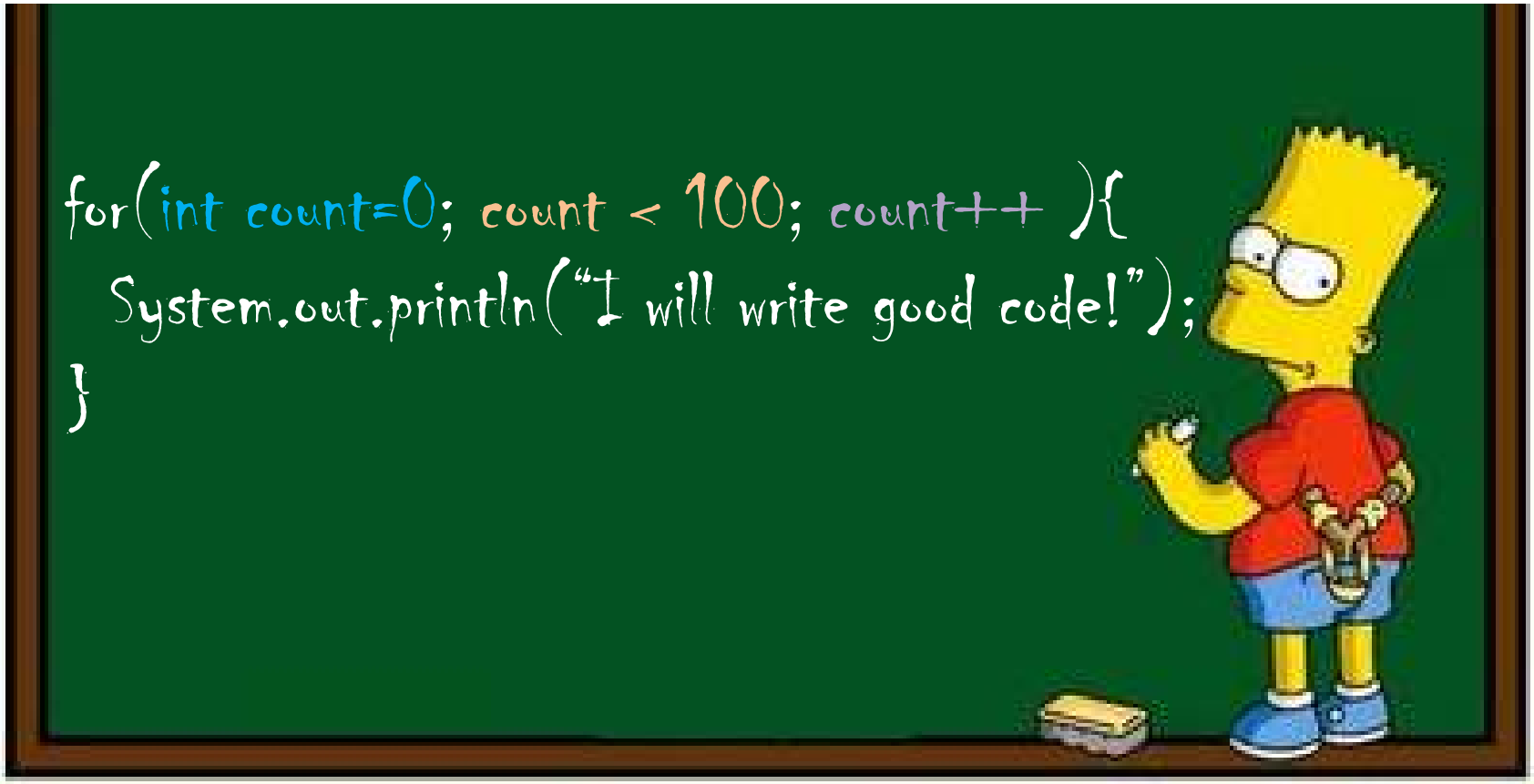
do-while Loop




```
do {
  // Loop body;
  Statement(s);
} while (loop-continuation-condition);
```

`for` Loops

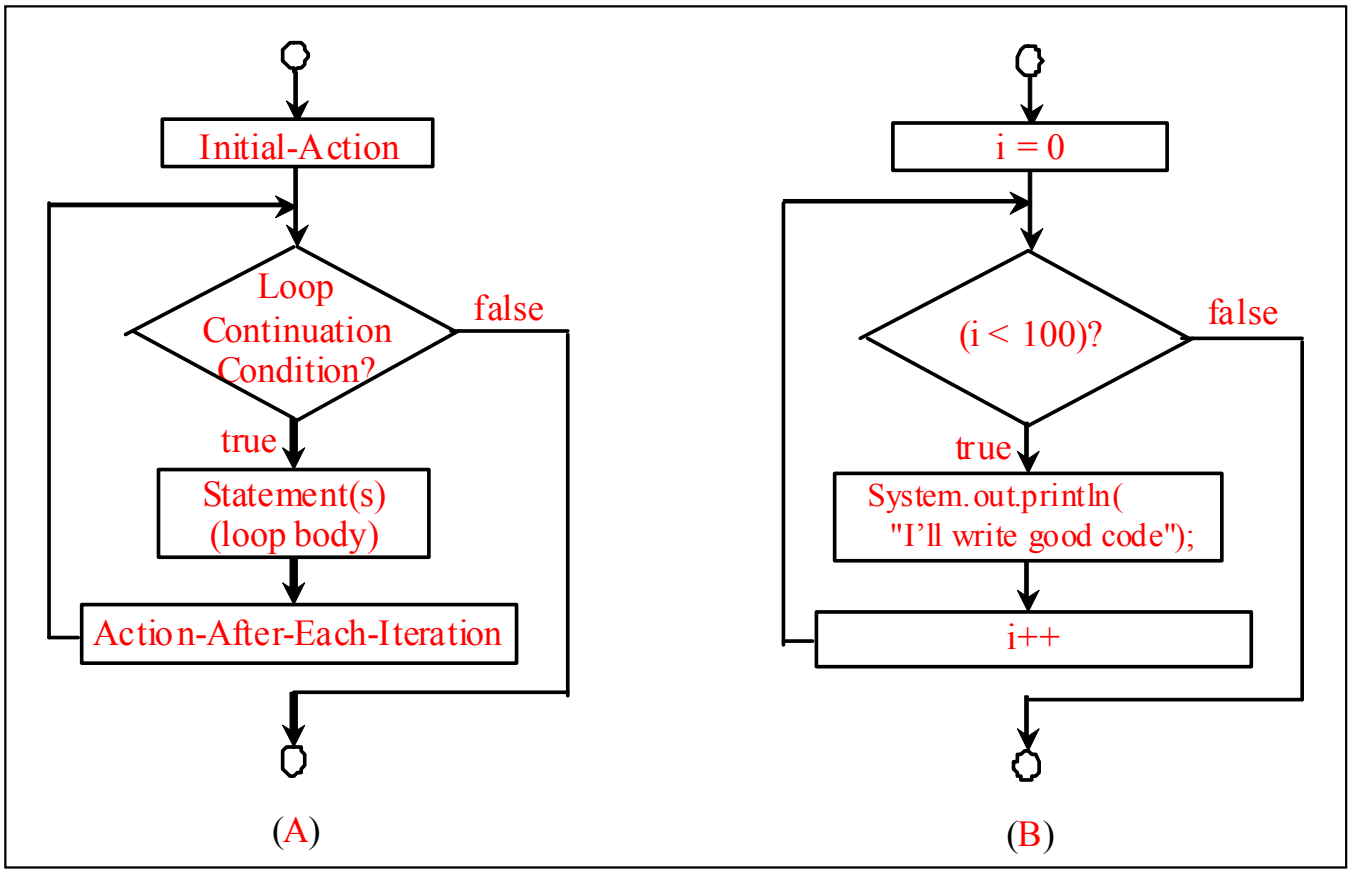
```
for (initial-action; loop-condition; action-after-each-iteration) {
   // loop body;
   Statement(s);
}
```



`for` Loops

```
for (initial-action; loop-
    continuation-condition;
    action-after-each-iteration) {
  // loop body;
  Statement(s);
}
```

```
for (int i = 0; i < 100; i++) {
  System.out.println(
    "I'll write good code!");
}
```



Which loop to use?

- Use the one that is most intuitive and comfortable for you.
- A for loop may be used if the number of repetitions is known, as, for example, when you need to print a message 100 times.
- A while loop may be used if the number of repetitions is not known, as in the case of reading the numbers until the input is 0.
- A do-while loop can be used to replace a while loop if the loop body has to be executed before testing the continuation condition.

Question

- What is the output of the following code fragment? _____.

```
int sum = 1;
for (int i = 0; i <= 5; sum = sum + i++);
System.out.print(sum);
```

Bonus question

What is the value of x after the following statements? ___

```
int x = 0, j = 0;
boolean done = false;
while(!done) {
   for (int i = 0; i<5; i++) {
      j = j + i;
      if (j > 12) {
         x = j;
         done = true;
      }
   }
}
```

Roadmap

- Lab session
- Pretest Postmortem
- Java Review
  - Types, variables, assignments, expressions
  - Control flow statements
  - Methods
  - Arrays
  - OO and Inheritance

Levels of Abstraction: Software Design

- Old times: computer programs manipulated primitive types such as numbers and characters
- Methods: Encapsulate routine computations to black boxes
- Object-oriented programming: Encapsulate data fields and methods to black boxes

Example – Computing sum

```
int sum = 0; int n=5;
for (int i = 1; i <= n; i++ ) {
    sum += i;
}
System.out.println("sum is:" + sum);
```

Example - Print the sums of 1 – 10, 25 – 35, 40 – 50

```
public static void main(String[] args) {
    int sum = 0;
    for (int i = 1; i <= 10; i++) {
            sum += i;
    }
    System.out.println("The sum of 1-10 is: " + sum);
    sum = 0;
    for (int i = 25; i <= 30; i++) {
            sum += i;
    }
    System.out.println("The sum of 25-30 is: " + sum);
    for (int i = 40; i <= 50; i++) {
            sum += i;
    }
    System.out.println("The sum of 40-50 is: " + sum);
}
```

Using a method *Sum*

If there was a method *sum* that would take two input integer arguments, *start* and *end*, and would add up and *return* sum of numbers from *start* to *end*

The program could be re-written much easier as:

```
public static void main(String[] args) {
    System.out.println("sum(1, 10) is: " + sum(1, 10) ); // 1+2+...+10
    System.out.println("sum(25, 30) is: " + sum(25, 30) ); //25+26+...+30
    System.out.println("sum(40, 50) is: " + sum(40, 50) ); //40+41+...+50
}
```

Defining a Method *sum*

```
public static int sum(int start, int end) {

    int sum = 0;
    for (int i = start; i <= end; i++) {
        sum += i;
    }
    return sum; // return is required
}
```

Defining and Using Methods

- Define a method – give a definition of what the method is to do

```
modifier returnType methodName(list of parameters) {
collection of statements;
}
```

- Call or invoke a method – use a method

```
methodName(list of parameters)
```