COS 126	Princeton University	Spring 2026

Written Exam

This exam has 10 questions worth a total of 100 points. You have 80 minutes to complete it.

Instructions. This exam is preprocessed by computer. Write neatly and legibly, using a dark pen or pencil. Put all answers (and nothing else) inside the designated spaces. *Fill in* bubbles and checkboxes completely (● and ■). To change an answer, erase it completely and rewrite it.

Resources. The exam is closed book, except that you may use a one-page reference sheet (8.5-by-11 paper, one side, handwritten by you). No electronic devices are permitted.

Honor Code. This exam is governed by Princeton's Honor Code. Discussing the contents of this exam before solutions are posted violates the Honor Code.

Please complete the following information:

Name:

NetID:

Exam room: ○ McCosh 50 ○ McCosh 66 ○ Other

Precept:

P01	P01A	P02	P03	P04	P04A	P05	P05A
○	○	○	○	○	○	○	○
P06	P10	P10A	P11	P12			
○	○	○	○	○			

"I pledge my honor that I will not violate the Honor Code during this examination."

Signature

1. **Initialization. (2 points)**

   In the designated spaces on the front of this exam,

   - *Write* your name.
   - *Write* your Princeton NetID (6–8 alphanumeric characters).
   - *Fill in* the bubble corresponding to the room in which you are taking this exam.
   - *Fill in* the bubble corresponding to your precept.
   - *Write* and *sign* the Honor Code pledge.

2. **Java expressions. (12 points)**

Assume the variables `w`, `x`, `y`, and `z` are initialized as follows:

```
boolean w = false;
int x = 5;
String y = "2";
double[] z = { 1.0, 2.0, 6.0 };
```

For each Java expression on the left, write the letter of the result on the right (a value, including its type; a compile-time error; or a runtime exception).

You may use each letter any number of times.

$\boxed{\text{H}}$ `1 + 1`

$\boxed{}$ `w || (x > 5) || (z[2] >= z[1])`

$\boxed{}$ `w <= x`

$\boxed{}$ `x + z[0]  /  z[1] - x`

$\boxed{}$ `z[0] <= z[1] <= z[2]`

$\boxed{}$ `Math.sqrt(x - 1)`

$\boxed{}$ `Integer.parseInt(x + y)`

$\boxed{}$ `(int) (x / 2.0)`

$\boxed{}$ `(double) (x / 2)`

A.	`false`
B.	`true`
C.	`-2.0`
D.	`0`
E.	`0.5`
F.	`1`
G.	`1.0`
H.	`2`
I.	`2.0`
J.	`2.5`
K.	`3`
L.	`3.0`
M.	`7`
N.	`52`
O.	*compile-time error*
P.	*runtime exception*

3. **Programming terminology. (10 points)**

For each Java term on the left, write the letter of the best-matching description on the right. Use each letter at most once.

- ☐ Side effect
- ☐ Runtime exception
- ☐ Parameter variable
- ☐ Local variable
- ☐ Constant variable
- ☐ Code block
- ☐ Data type
- ☐ Array
- ☐ Encapsulation
- ☐ Classpath

A. The locations where Java looks for user-defined libraries and classes

B. A set of values together with operations on those values

C. A variable whose value cannot change

D. A variable declared inside a method (including inside loops or conditionals)

E. A variable listed in a method header that receives a value when the method is called

F. An observable action a function performs other than computing and returning a value

G. A set of related functions stored in a single file, intended for use by other programs

H. A sequence of values of the same type, stored in order

I. A sequence of statements enclosed in curly braces that Java treats as a single unit

J. A named sequence of statements you can call from elsewhere

K. Hiding implementation details behind a well-defined API

L. An error reported by the compiler for an invalid Java program

M. An error that occurs while the program is running, which may terminate the program

4. **Standard input and command-line arguments. (12 points)**

Consider `Reverse.java` and `Shift.java`, along with the text file `input.txt`:

```
public class Reverse {
   public static void main(String[] args) {
      for (int i = args.length - 1; i >= 0; i--)
         StdOut.println(args[i]);
   }
}
```

```
public class Shift {
   public static void main(String[] args) {
      int k = Integer.parseInt(args[0]);
      String[] a = StdIn.readAllStrings();
      for (int i = 0; i < a.length; i++)
         StdOut.println(a[(i + k) % a.length]);
   }
}
```

Note: `StdIn.readAllStrings()` *reads all remaining strings from standard input and returns them as an array.*

```
% more input.txt
B A C
```

For each command on the left, determine the terminal output (ignoring newlines). Choose the best-matching letter on the right. You may use each letter any number of times.

☐	`java-introcs Reverse C B A`	**A.**	`A B C`
☐	`java-introcs Reverse A B C > output.txt`	**B.**	`A C B`
☐	`java-introcs Reverse < input.txt`	**C.**	`B A C`
☐	`java-introcs Shift < input.txt`	**D.**	`B C A`
☐	`java-introcs Shift 1 < input.txt`	**E.**	`C A B`
☐	`java-introcs Reverse A B C | java-introcs Shift 2`	**F.**	`C B A`
☐	`java-introcs Shift 10 < input.txt | java-introcs Shift 20`	**G.**	*no output in the terminal*
		H.	*runtime exception*

5. **Java language properties. (10 points)**

Identify each statement about Java as either true or false by filling in the appropriate bubble.

true	*false*	
○	○	A single array can store both an element of type `int` and an element of type `String`.
○	○	Once you create an array, you cannot change that array's length.
○	○	A `for` loop's update expression always executes at least once, even if the loop body does not execute.
○	○	A variable declared in a `for` loop header can be accessed outside that `for` loop.
○	○	The expression `a[-1]` refers to the last element of the array `a[]`.
○	○	Passing an `int` to a function that expects a `String` produces a compile-time error.
○	○	Passing an `int[]` to a function that expects a `double[]` produces a compile-time error.
○	○	A `private` function can be called by any function in the same class, including `main()`.
○	○	Every variable has a declared type.
○	○	Attempting to assign a value to a variable of an incompatible type produces a compile-time error.

6. **Loops, conditionals, and debugging. (10 points)**

   Each code fragment below attempts to read numbers from standard input and assign to `result` the value closest to `target`. For example, if `target` is `3.0` and standard input is `10.0 2.5 -2.0 8.0`, then `result` should be `2.5`.

   Mark each code fragment as correct or incorrect. Assume that standard input contains at least one value and that `target` *is an initialized variable of type* `double`*.*

correct *incorrect*

○ ○

```
double result = StdIn.readDouble();
double bestDelta = Math.abs(result - target);
while (!StdIn.isEmpty()) {
    double x = StdIn.readDouble();
    double delta = Math.abs(x - target);
    if (delta < bestDelta) {
        result = x;
        bestDelta = delta;
    }
}
```

○ ○

```
double result = Double.POSITIVE_INFINITY;
while (!StdIn.isEmpty()) {
    double x = StdIn.readDouble();
    if (Math.abs(target - x) < Math.abs(result - x))
        result = x;
}
```

○ ○

```
double result = 0.0;
double bestDelta = Double.POSITIVE_INFINITY;
while (!StdIn.isEmpty()) {
    double x = StdIn.readDouble();
    double delta = Math.abs(x - target);
    bestDelta = Math.min(delta, bestDelta);
    if (delta < bestDelta)
        result = x;
}
```

7. **Conditionals and functions. (10 points)**

For each function below, write the letter of the best-matching description from the choices below. You may use each letter any number of times.

A. Returns `true` when **all** of its inputs are `true`; `false` otherwise.

B. Returns `true` when **at least one** of its inputs are `true`; `false` otherwise.

C. Returns `true` when **at least two** of its inputs are `true`; `false` otherwise.

D. Returns `true` when an **odd number** of its inputs are `true`; `false` otherwise.

```
public static boolean f1(boolean x, boolean y, boolean z) {
   if (x && y) return true;
   if (x && z) return true;
   return y && z;
}
```

```
public static boolean f2(boolean x, boolean y, boolean z) {
   if (x) return true;
   else if (y) return true;
   else if (z) return true;
   else return false;
}
```

```
public static boolean f3(boolean x, boolean y, boolean z) {
   int count = 0;
   if (x) count++;
   if (y) count++;
   if (z) count++;
   return count % 2 != 0;
}
```

```
public static boolean f4(boolean x, boolean y, boolean z) {
   if (x && y) return z;
   if (x || y) return false;
   return x && z;
}
```

8. **Functions, arrays, and pass-by-value. (10 points)**

Consider the following Java functions:

```
public static int twist(int x, int y) {
    x = y;
    return x;
}

public static void twistOne(int[] a, int[] b) {
    int n = a.length;
    for (int i = 0; i < n; i++)
        a[i] = b[n - i - 1];
    for (int i = 0; i < n; i++)
        b[i] = a[n - i - 1];
}

public static void twistTwo(int[] a, int[] b) {
    int n = a.length;
    for (int i = 0; i < n; i++)
        twist(a[i], b[n - i - 1]);
    for (int i = 0; i < n; i++)
        twist(b[i], a[n - i - 1]);
}
```

Before each part, assume the three `int[]` arrays are initialized as follows: `a = { 1, 2, 3 }`, `b = { 4, 5, 6 }`, and `c = { 7, 8 }`. After executing the statement, what is `a[]`?

For each statement on the left, write the letter of the best-matching description on the right. You may use each letter any number of times.

☐	`twistOne(a, b);`	**A.** `{ 1, 2, 1 }`
		B. `{ 1, 2, 3 }`
☐	`twistTwo(a, b);`	**C.** `{ 2, 4, 6 }`
		D. `{ 3, 2, 1 }`
☐	`twistOne(a, a);`	**E.** `{ 3, 2, 3 }`
		F. `{ 4, 5, 6 }`
		G. `{ 6, 5, 4 }`
☐	`twistOne(a, c);`	**H.** `{ 8, 7, 3 }`
		I. *runtime exception*

9. **Recursion. (12 points)**

Consider the following recursive function:

```
public static String f(int n) {
   if (n <= 0) return "A";                          // line 1
   if (n == 1) return "B";                          // line 2
   String first  = f(n - 1);                        // line 3
   String second = f(n - 2);                        // line 4
   String third  = f(n - 1);                        // line 5
   return first + "C" + second + "C" + third;       // line 6
}
```

Mark each statement as either *true* or *false*.

true	*false*	
●	○	`f(1)` returns `"B"`.
○	○	`f(2)` returns `"BCACB"`.
○	○	`f(3)` returns `"BCACBCACBCACB"`.
○	○	The length (number of characters) of `f(5)` is ≥ 67.
○	○	If line 1 is removed, then calling `f(10)` results in a runtime exception (e.g., `StackOverflowError` or `OutOfMemoryError`).
○	○	Swapping lines 3 and 4 does not change the string returned by `f(n)`.
○	○	Any string returned by `f(n)` is a *palindrome* (it reads the same forward and backward).

10. **Performance. (12 points)**

Determine the *order of growth of the running time* of each of the following functions as a function of n.

For each function on the left, write the letter of the best-matching term on the right. You may use each letter any number of times.

```
public static int f1(int n) {
    int count = 0;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            count++;
    return count;
}
```

```
public static int f2(int n) {
    int count = 0;
    for (int i = n; i >= 1; i--)
        for (int j = n; j >= 1; j = j / 2)
            count++;
    return count;
}
```

```
public static int f3(int n) {
    if (n == 0) return 0;
    return 1 + f3(n - 1);
}
```

```
public static int f4(int n) {
    int count = 0;
    for (int i = 0; i < n*n; i++)
        count++;
    for (int i = 0; i*i < f1(n); i++)
        count += f1(n);
    return count;
}
```

A. $\Theta(1)$ *constant*

B. $\Theta(\log n)$ *logarithmic*

C. $\Theta(\sqrt{n})$ *square root*

D. $\Theta(n)$ *linear*

E. $\Theta(n \log n)$ *linearithmic*

F. $\Theta(n^2)$ *quadratic*

G. $\Theta(n^{2.5})$

H. $\Theta(n^3)$ *cubic*

I. $\Theta(n^4)$ *quartic*

J. $\Theta(n^5)$ *quintic*

K. $\Theta(2^n)$ *exponential*