Introduction to Java

m(ethods)

A Modern Smartphone

A Modern Smartphone

- Surf the net
  - Input: Web address
  - Output: Desired page

A Modern Smartphone

- Surf the net
  - Input: Web address
  - Output: Desired page
- Book tickets
  - Input: userid, password, booking info, bank info
  - Output: Ticket

A Modern Smartphone

- Surf the net
  - Input: Web address
  - Output: Desired page
- Book tickets
  - Input: userid, password, booking info, bank info
  - Output: Ticket
- Send email
  - Input: email address of receiver, mail text
  - Output: --

A Modern Smartphone

- Surf the net
  - Input: Web address
  - Output: Desired page
- Book tickets
  - Input: userid, password, booking info, bank info
  - Output: Ticket
- Send email
  - Input: email address of receiver, mail text
  - Output: --
- Take photos
  - Input: --
  - Output: Picture

A Modern Smartphone

- Surf the net
  - Input: Web address
  - Output: Desired page
- Book tickets
  - Input: userid, password, booking info, bank info
  - Output: Ticket
- Send email
  - Input: email address of receiver, mail text
  - Output: --
- Take photos
  - Input: --
  - Output: Picture
- Talk (we can do that too!!)
  - Input: Phone number
  - Output: Conversation (if lucky)

A Modern Smartphone

- Surf the net
  - Input: Web address
  - Output: Desired page
- Book tickets
  - Input: userid, password, booking info, bank info
  - Output: Ticket
- Send email
  - Input: email address of receiver, mail text
  - Output: --
- Take photos
  - Input: --
  - Output: Picture
- Talk (we can do that too!!)
  - Input: Phone number
  - Output: Conversation (if lucky)
- …

Lots of related/unrelated task to perform

Lots of related/unrelated task to perform

- Divide and Conquer

Lots of related/unrelated task to perform

- Divide and Conquer
  - Create well defined sub tasks

Lots of related/unrelated task to perform

- Divide and Conquer
  - Create well defined sub tasks
  - Work on each task independently

Lots of related/unrelated task to perform

- Divide and Conquer
  - Create well defined sub tasks
  - Work on each task independently
    - Development, Enhancements, Debugging

Lots of related/unrelated task to perform

- Divide and Conquer
  - Create well defined sub tasks
  - Work on each task independently
    - Development, Enhancements, Debugging
- Reuse of tasks.

Lots of related/unrelated task to perform

- Divide and Conquer
  - Create well defined sub tasks
  - Work on each task independently
    - Development, Enhancements, Debugging
- Reuse of tasks.
  - Email and Chat apps can share spell checker.

Lots of related/unrelated task to perform

- Divide and Conquer
  - Create well defined sub tasks
  - Work on each task independently
    - Development, Enhancements, Debugging
- Reuse of tasks.
  - Email and Chat apps can share spell checker.
  - Phone and SMS apps can share dialer

Lots of related/unrelated task to perform

- Divide and Conquer
  - Create well defined sub tasks
  - Work on each task independently
    - Development, Enhancements, Debugging
- Reuse of tasks.
  - Email and Chat apps can share spell checker.
  - Phone and SMS apps can share dialer
- Can be facilitated using methods

Method

Method

- An independent, self-contained entity of a program that performs a well-defined task.

Method

- An independent, self-contained entity of a program that performs a well-defined task.
- It has
  - Name: for identification
  - Arguments: to pass information from outside world (rest of the program)
  - Body: processes the arguments do something useful
  - Return value: To communicate back to outside world
    - Sometimes not required

Why use Methods?

Example : Maximum of 3 numbers

```
… main(…){
   int a, b, c, m;

   /* code to read
    * a, b, c */

   if (a>b){
     if (a>c) m = a;
     else m = c;
   }
   else{
     if (b>c) m = b;
     else m = c;
   }

   /* print or use m */


}
```

Why use Methods?

Example : Maximum of 3 numbers

```
… main(…){
   int a, b, c, m;

   /* code to read
    * a, b, c */

   if (a>b){
      if (a>c) m = a;
      else m = c;
   }
   else{
      if (b>c) m = b;
      else m = c;
   }

   /* print or use m */


}
```

Why use Methods?

Example : Maximum of 3 numbers

```
… main(…){
   int a, b, c, m;

   /* code to read
    * a, b, c */

   if (a>b){
     if (a>c) m = a;
     else m = c;
   }
   else{
     if (b>c) m = b;
     else m = c;
   }

   /* print or use m */


}
```

Why use Methods?

Example : Maximum of 3 numbers

```
… main(…){
   int a, b, c, m;

   /* code to read
    * a, b, c */

   if (a>b){
     if (a>c) m = a;
     else m = c;
   }
   else{
     if (b>c) m = b;
     else m = c;
   }

   /* print or use m */


}
```

Why use Methods?

Example : Maximum of 3 numbers

```
… main(…){
   int a, b, c, m;

   /* code to read
    * a, b, c */

   if (a>b){
     if (a>c) m = a;
     else m = c;
   }
   else{
     if (b>c) m = b;
     else m = c;
   }

   /* print or use m */


}
```

```
int max(int a, int b){
   if (a>b)
     return a;
   else
     return b;
}

… main(…) {
   int a, b, c, m;

   /* code to read
    * a, b, c */

   m = max(a, b);
   m = max(m, c);
   /* print or use m */


}
```

Why use Methods?

Example : Maximum of 3 numbers

```
… main(…){
   int a, b, c, m;

   /* code to read
    * a, b, c */

   if (a>b){
     if (a>c) m = a;
     else m = c;
   }
   else{
     if (b>c) m = b;
     else m = c;
   }

   /* print or use m */


}
```

```
int max(int a, int b){
   if (a>b)
     return a;
   else
     return b;
}

… main(…) {
   int a, b, c, m;

   /* code to read
    * a, b, c */

   m = max(a, b);
   m = max(m, c);
   /* print or use m */


}
```

This code can scale easily to handle large number of inputs (e.g.: max of 100 numbers!)

Why use Methods?

Why use Methods?

- Break up complex problem into small sub-problems.

Why use Methods?

- Break up complex problem into small sub-problems.
- Solve each of the sub-problems separately as a method, and combine them together in another method.

Why use Methods?

- Break up complex problem into small sub-problems.
- Solve each of the sub-problems separately as a method, and combine them together in another method.
- The main tool for modular programming.

We have seen Methods before

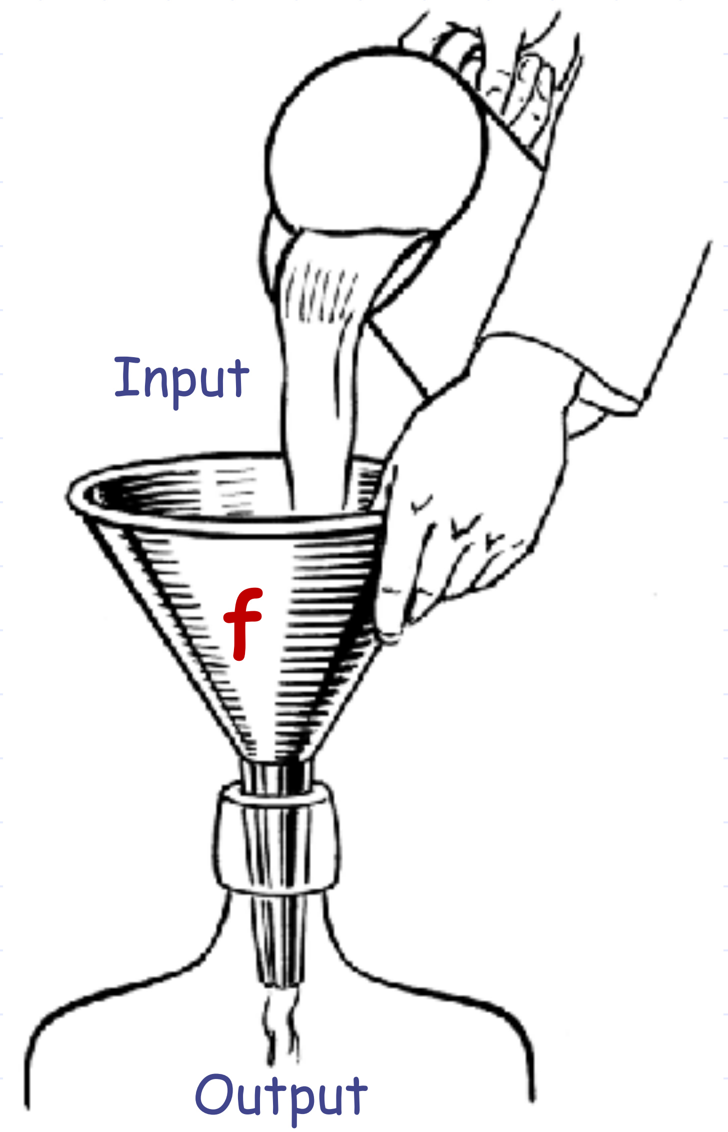
We have seen Methods before

- main() is a special method. Execution of program starts from the beginning of main().

We have seen Methods before

- main() is a special method. Execution of program starts from the beginning of main().
- println(…), read(…) are standard input-output library methods.

Parts of a method



```
int  max (int a, int b) {
   if (a > b)
       return a;
  else
       return b;
}
```

```
... main (...) {
     int x;
     x = max(6, 4);
     println("max "+x);

}
```

Return Type

```
int max (int a, int b) {
   if (a > b)
      return a;
  else
      return b;
}
```

```
… main (…) {
     int x;
     x = max(6, 4);
     println("max "+x);

}
```

```
int max (int a, int b) {
   if (a > b)
       return a;
  else
       return b;
}
```

Return Type

Method Name

```
... main (...) {
      int x;
      x = max(6, 4);
      println("max "+x);

}
```

Return Type

Method Name

```
int max (int a, int b) {
   if (a > b)
      return a;
  else
      return b;
}
```

2 arguments
a and b,
both of type int.
(formal args)

```
... main (...) {
      int x;
      x = max(6, 4);
      println("max "+x);

}
```

```
int max (int a, int b) {
   if (a > b)
      return a;
   else
      return b;
}
```

Return Type

Method Name

2 arguments
a and b,
both of type int.
(formal args)

```
... main (...) {
      int x;
      x = max(6, 4);
      println("max "+x);

}
```

Body of the method, enclosed inside { and } (mandatory) returns an int.

```
int max (int a, int b) {
    if (a > b)
        return a;
    else
        return b;
}
```

Return Type

Method Name

2 arguments
a and b,
both of type int.
(formal args)

```
... main (...) {
        int x;
        x = max(6, 4);
        println("max "+x);

}
```

Body of the
method, enclosed
inside { and }
(mandatory)
returns an int

Call to the method.
Actual args are 6 and 4.

Method Call

Method Call

- A method call is an *expression*
  - feeds the necessary values to the method arguments,
  - directs a method to perform its task, and
  - receives the return value of the method.

Method Call

- A method call is an *expression*
  - feeds the necessary values to the method arguments,
  - directs a method to perform its task, and
  - receives the return value of the method.
- Similar to operator application

Method Call

- A method call is an *expression*
  - feeds the necessary values to the method arguments,
  - directs a method to perform its task, and
  - receives the return value of the method.
- Similar to operator application

5 + 3 is an expression of type integer that evaluates to 8

Method Call

- A method call is an *expression*
  - feeds the necessary values to the method arguments,
  - directs a method to perform its task, and
  - receives the return value of the method.
- Similar to operator application

5 + 3 is an expression of type integer that evaluates to 8

max(5, 3) is an expression of type integer that evaluates to 5

Method Call

Method Call

- Since a method call is an *expression*
  - it can be used anywhere an expression can be used
  - subject to type restrictions

Method Call

- Since a method call is an *expression*
  - it can be used anywhere an expression can be used
  - subject to type restrictions

```
println(""+max(5,3));
max(5,3) – min(5,3)
max(x, max(y, z)) == z

if (max(a, b)!=0)
            println("Y");
```

Method Call

- Since a method call is an *expression*
  - it can be used anywhere an expression can be used
  - subject to type restrictions

```
println(“”+max(5,3));
max(5,3) – min(5,3)
max(x, max(y, z)) == z

if (max(a, b)!=0)
            println(“Y”);
```

```
prints 5
```

Method Call

- Since a method call is an *expression*
  - it can be used anywhere an expression can be used
  - subject to type restrictions

```
println(“”+max(5,3));
max(5,3) – min(5,3)
max(x, max(y, z)) == z

if (max(a, b)!=0)
                println(“Y”);
```

prints 5

evaluates to 2

Method Call

- Since a method call is an *expression*
  - it can be used anywhere an expression can be used
  - subject to type restrictions

`println(""+max(5,3));`	prints 5
`max(5,3) – min(5,3)`	evaluates to 2
`max(x, max(y, z)) == z`	checks if z is max of x, y, z

```
if (max(a, b)!=0)
              println("Y");
```

Method Call

- Since a method call is an *expression*
  - it can be used anywhere an expression can be used
  - subject to type restrictions

`println(""+max(5,3));`	prints 5
`max(5,3) – min(5,3)`	evaluates to 2
`max(x, max(y, z)) == z`	checks if z is max of x, y, z
`if (max(a, b)!=0) println("Y");`	prints Y if max of a and b is not 0.

Returning from a method: Type

Returning from a method: Type

- Return type of a method tells the type of the result of method call

Returning from a method: Type

- Return type of a method tells the type of the result of method call
- Any valid Java type
  - int, char, float, double, …
  - **void**

Returning from a method: Type

- Return type of a method tells the type of the result of method call
- Any valid Java type
  - int, char, float, double, …
  - **void**
- Return type is void if the method is not supposed to return any value

Returning from a method: Type

- Return type of a method tells the type of the result of method call
- Any valid Java type
  - int, char, float, double, …
  - **void**
- Return type is void if the method is not supposed to return any value

```
void print_one_int(int n) {
    println(“”+ n);
}
```

Returning from a method: return statement

Returning from a method: return statement

- If return type is not void, then the method MUST return a value:

Returning from a method: return statement

- If return type is not void, then the method MUST return a value:

```
return return_expr;
```

Returning from a method: return statement

- If return type is not void, then the method MUST return a value:

  return return_expr;

- If return type is void, the method may *fall through* at the end of the body or use a return without return_expr:

Returning from a method: return statement

- If return type is not void, then the method MUST return a value:

  return return_expr;

- If return type is void, the method may *fall through* at the end of the body or use a return without return_expr:

  return;

Returning from a method: return statement

- If return type is not void, then the method MUST return a value:

  return return_expr;

- If return type is void, the method may *fall through* at the end of the body or use a return without return_expr:

  return;

```
void print_positive(int n) {
    if (n <= 0) return;
    println(""+n);
}
```

Returning from a method: return statement

- If return type is not void, then the method MUST return a value:

    return return_expr;

- If return type is void, the method may *fall through* at the end of the body or use a return without return_expr:

    return;

```
void print_positive(int n) {
    if (n <= 0) return;
    println(""+n);
}
```

Returning through return

Returning from a method: return statement

- If return type is not void, then the method MUST return a value:

  return return_expr;

- If return type is void, the method may *fall through* at the end of the body or use a return without return_expr:

  return;

```
void print_positive(int n) {
    if (n <= 0) return;
    println(""+n);
}
```

Returning through return

Fall through

Returning from a method: return statement

Returning from a method: return statement

- When a return statement is encountered in a method definition
  - control is immediately transferred back to the statement making the method call in the parent method.

Returning from a method: return statement

- When a return statement is encountered in a method definition
  - control is immediately transferred back to the statement making the method call in the parent method.
- A method in Java can return only ONE value or NONE.
  - Only one return type (including void)

Nested Method Calls

Nested Method Calls

- Methods can call each other

Nested Method Calls

- Methods can call each other

```
int max(int a, int b) {
```

Nested Method Calls

- Methods can call each other

```
int max(int a, int b) {
  return (a > b) ? a : b;
```

Nested Method Calls

- Methods can call each other

```
int max(int a, int b) {
  return (a > b) ? a : b;
}
```

Nested Method Calls

- Methods can call each other

```
int max(int a, int b) {
   return (a > b) ? a : b;
}
```

Nested Method Calls

- Methods can call each other

```
int max(int a, int b) {
  return (a > b) ? a : b;
}

// a “cryptic” min, uses max
```

Nested Method Calls

- Methods can call each other

```
int max(int a, int b) {
   return (a > b) ? a : b;
}

// a "cryptic" min, uses max
int min(int a, int b) {
```

Nested Method Calls

- Methods can call each other

```
int max(int a, int b) {
   return (a > b) ? a : b;
}

// a "cryptic" min, uses max
int min(int a, int b) {
   return a + b – max (a, b);
```

Nested Method Calls

- Methods can call each other

```
int max(int a, int b) {
   return (a > b) ? a : b;
}

// a “cryptic” min, uses max
int min(int a, int b) {
   return a + b – max (a, b);
}
```

Nested Method Calls

- Methods can call each other

```
int max(int a, int b) {
   return (a > b) ? a : b;
}

// a “cryptic” min, uses max
int min(int a, int b) {
   return a + b – max (a, b);
}
```

Nested Method Calls

- Methods can call each other

```
int max(int a, int b) {
   return (a > b) ? a : b;
}

// a “cryptic” min, uses max
int min(int a, int b) {
   return a + b – max (a, b);
}

… main(…) {
```

Nested Method Calls

- Methods can call each other

```
int max(int a, int b) {
   return (a > b) ? a : b;
}

// a "cryptic" min, uses max
int min(int a, int b) {
   return a + b – max (a, b);
}

… main(…) {
  println(""+min(6, 4));
```

Nested Method Calls

- Methods can call each other

```
int max(int a, int b) {
   return (a > b) ? a : b;
}

// a “cryptic” min, uses max
int min(int a, int b) {
   return a + b – max (a, b);
}

… main(…) {
  println(“”+min(6, 4));
}
```

Predefined Methods

Predefined Methods

- Java has many predefined methods as part of standard library.

Predefined Methods

- Java has many predefined methods as part of standard library.
- We have seen System.out.println.

Predefined Methods

- Java has many predefined methods as part of standard library.
- We have seen System.out.println.
- To use a predefined method, the corresponding library must be imported of full name must be used.

Predefined Methods

- Java has many predefined methods as part of standard library.
- We have seen System.out.println.
- To use a predefined method, the corresponding library must be imported of full name must be used.
  - Mathematical Methods like sin, cos, log, pow etc. defined in the library java.lang.Math

Predefined Methods

- Java has many predefined methods as part of standard library.
- We have seen System.out.println.
- To use a predefined method, the corresponding library must be imported of full name must be used.
  - Mathematical Methods like sin, cos, log, pow etc. defined in the library java.lang.Math
  - Input/Output Methods in System.in/ System.out

Predefined Methods

- Java has many predefined methods as part of standard library.
- We have seen System.out.println.
- To use a predefined method, the corresponding library must be imported of full name must be used.
  - Mathematical Methods like sin, cos, log, pow etc. defined in the library java.lang.Math
  - Input/Output Methods in System.in/ System.out
  - Consult the documentation

Avoiding Common Errors

Avoiding Common Errors

- Argument list of a method:
  - Provide the required number of arguments,
  - Check that each method argument has the correct type (or that conversion to the correct type will lose no information).

Avoiding Common Errors

- Return statement
  - value must match the return type
  - return statement must be encountered during execution for a function having non void return type

Avoiding Common Errors

- Return statement
  - value must match the return type
  - return statement must be encountered during execution for a function having non void return type
- Also be careful in using functions that are undefined on some values.
  - $\sin^{-1}(x)$ is defined only for $-1 \le x \le 1$
  - In Java double asin(double x)
    - pronounced a-sine or arc-sine

Scope of a Name

Scope of a Name

- Methods allow us to divide a program into smaller parts
    - each part does a well defined task

Scope of a Name

- Methods allow us to divide a program into smaller parts
  - each part does a well defined task
- There are other ways to *partition a program*
  - *Statement blocks, Files*

Scope of a Name

- Methods allow us to divide a program into smaller parts
  - each part does a well defined task
- There are other ways to *partition a program*
  - *Statement blocks, Files*
- Scope of a name is the part of the program in which the name can be used

Scope of a Name

Scope of a Name

- Two variables can have the same name only if they are declared in separate scopes.
  - Java does not allow **shadowing**
  - A name cannot be redeclared in a nested scope.

Scope of a Name

- Two variables can have the same name only if they are declared in separate scopes.
    - Java does not allow **shadowing**
    - A name cannot be redeclared in a nested scope.
- A variable can not be used outside its scope.

Scope of a Name

- Two variables can have the same name only if they are declared in separate scopes.
  - Java does not allow **shadowing**
  - A name cannot be redeclared in a nested scope.
- A variable can not be used outside its scope.
- Java program has
  - method/block scope
  - Class scope
  - Static scope

Scope Rules: Methods

Scope Rules: Methods

- The scope of the variables present in the argument list of a method's definition is the body of that method.

Scope Rules: Methods

- The scope of the variables present in the argument list of a method's definition is the body of that method.
- The scope of any variable declared within a method is the body of the method.

Scope Rules: Methods

- The scope of the variables present in the argument list of a method's definition is the body of that method.
- The scope of any variable declared within a method is the body of the method.

```
int max(int a1, int b1) {
   int m1 = 0;
   if (a1 > b1) m1 = a1;
   else m1 = b1;
   return m1;
}

int min(int a2, int b2) {
   int m2 = 0;
   if (a2 < b2) m2 = a2;
   else m2 = b2;
   return m2;
}
```

Scope Rules: Methods

- The scope of the variables present in the argument list of a method's definition is the body of that method.
- The scope of any variable declared within a method is the body of the method.

scope of m1, a1, b1

```
int max(int a1, int b1) {
   int m1 = 0;
   if (a1 > b1) m1 = a1;
   else m1 = b1;
   return m1;
}

int min(int a2, int b2) {
   int m2 = 0;
   if (a2 < b2) m2 = a2;
   else m2 = b2;
   return m2;
}
```

Scope Rules: Methods

- The scope of the variables present in the argument list of a method's definition is the body of that method.
- The scope of any variable declared within a method is the body of the method.

scope of m1, a1, b1

```
int max(int a1, int b1) {
    int m1 = 0;
    if (a1 > b1) m1 = a1;
    else m1 = b1;
    return m1;
}
```

scope of m2, a2, b2

```
int min(int a2, int b2) {
    int m2 = 0;
    if (a2 < b2) m2 = a2;
    else m2 = b2;
    return m2;
}
```

Scope Rules: Methods

- The scope of the variables present in the argument list of a method's definition is the body of that method.
- The scope of any variable declared within a method is the body of the method.

```
int max(int a, int b) {
   int m = 0;
   if (a > b) m = a;
   else m = b;
   return m;
}

int min(int a, int b) {
   int m = 0;
   if (a < b) m = a;
   else m = b;
   return m;
}
```

Quiz: Argument Passing

```
// swapping a and b
void swap(int a, int b){
   int temp;
   temp = a;
   a = b;
   b = temp;
   printf("a=%d b=%d\n", a, b);
}
public static void main(…){
   int  a=10, b=15;
   printf("a=%d b=%d\n", a, b);
   swap(a, b);
   printf("a=%d b=%d\n", a, b);

}
```

What is the output of the program?
(fill the blanks)

OUTPUT

a=____ b=____

a=____ b=____

a=____ b=____

Quiz: Argument Passing

```
// swapping a and b
void swap(int a, int b){
   int temp;
   temp = a;
   a = b;
   b = temp;
   printf("a=%d b=%d\n", a, b);
}
public static void main(…){
   int  a=10, b=15;
   printf("a=%d b=%d\n", a, b);
   swap(a, b);
   printf("a=%d b=%d\n", a, b);

}
```

What is the output of the program? (fill the blanks)

OUTPUT

a=____ b=____

a=____ b=____

a=____ b=____

Quiz: Argument Passing

```
// swapping a and b
void swap(int a, int b){
   int temp;
   temp = a;
   a = b;
   b = temp;
   printf("a=%d b=%d\n", a, b);
}
public static void main(…){
   int  a=10, b=15;
   printf("a=%d b=%d\n", a, b);
   swap(a, b);
   printf("a=%d b=%d\n", a, b);

}
```

What is the output of the program? (fill the blanks)

OUTPUT

a=_10_ b=_15_

a=____ b=____

a=____ b=____

Quiz: Argument Passing

```
// swapping a and b
void swap(int a, int b){
   int temp;
   temp = a;
   a = b;
   b = temp;
   printf("a=%d b=%d\n", a, b);
}
public static void main(…){
   int  a=10, b=15;
   printf("a=%d b=%d\n", a, b);
   swap(a, b);
   printf("a=%d b=%d\n", a, b);

}
```

What is the output of the program? (fill the blanks)

OUTPUT

a=_10_ b=_15_

a=____ b=____

a=____ b=____

Quiz: Argument Passing

```
// swapping a and b
void swap(int a, int b){
   int temp;
   temp = a;
   a = b;
   b = temp;
   printf("a=%d b=%d\n", a, b);
}
public static void main(…){
   int  a=10, b=15;
   printf("a=%d b=%d\n", a, b);
   swap(a, b);
   printf("a=%d b=%d\n", a, b);

}
```

What is the output of the program? (fill the blanks)

OUTPUT

a= 10 b= 15

a= 15 b= 10

a=____ b=____

Quiz: Argument Passing

```
// swapping a and b
void swap(int a, int b){
  int temp;
  temp = a;
  a = b;
  b = temp;
  printf("a=%d b=%d\n", a, b);
}
public static void main(…){
  int  a=10, b=15;
  printf("a=%d b=%d\n", a, b);
  swap(a, b);
  printf("a=%d b=%d\n", a, b);

}
```

What is the output of the program? (fill the blanks)

OUTPUT

a=<u>10</u> b=<u>15</u>

a=<u>15</u> b=<u>10</u>

a=____ b=____

Quiz: Argument Passing

```
// swapping a and b
void swap(int a, int b){
   int temp;
   temp = a;
   a = b;
   b = temp;
   printf("a=%d b=%d\n", a, b);
}
public static void main(…){
   int  a=10, b=15;
   printf("a=%d b=%d\n", a, b);
   swap(a, b);
   printf("a=%d b=%d\n", a, b);

}
```

What is the output of the program?
(fill the blanks)

OUTPUT

```
a= 10  b= 15

a= 15  b= 10

a= 10  b= 15
```

Quiz: Argument Passing

```
// swapping a and b
void swap(int a, int b){
   int temp;
   temp = a;
   a = b;
   b = temp;
   printf("a=%d b=%d\n", a, b);
}
public static void main(…){
   int  a=10, b=15;
   printf("a=%d b=%d\n", a, b);
   swap(a, b);
   printf("a=%d b=%d\n", a, b);

}
```

What is the output of the program? (fill the blanks)

OUTPUT

a=_10_ b=_15_

a=_15_ b=_10_

a=_10_ b=_15_