Deutsche Bank
Corporate Division

Day 1

Introduction to JAVA

PPT-Master Version 01. There will be an extensive update with the new Deutsche Bank font and sample charts at the end of August 2024.

8 May 2024, Speaker name

INTRODUCTION

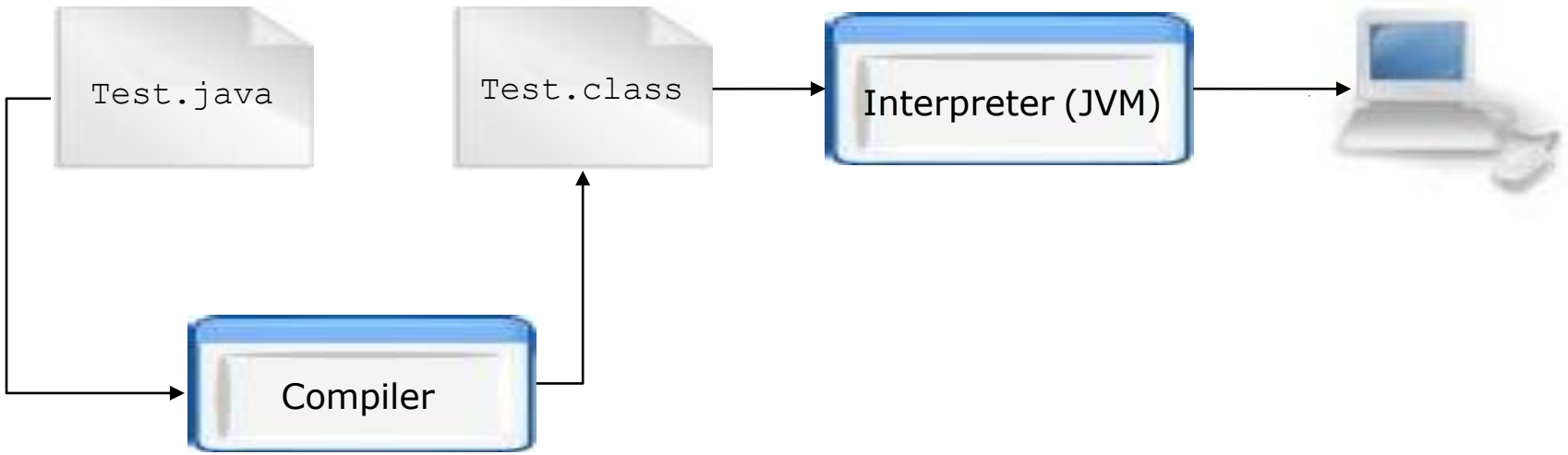
— The world wide web has popularized the use of Java, because programs can be transparently downloaded with web pages and executed in any computer with a Java capable browser.

— A Java application is a standalone Java program that can be executed independently of any web browser.

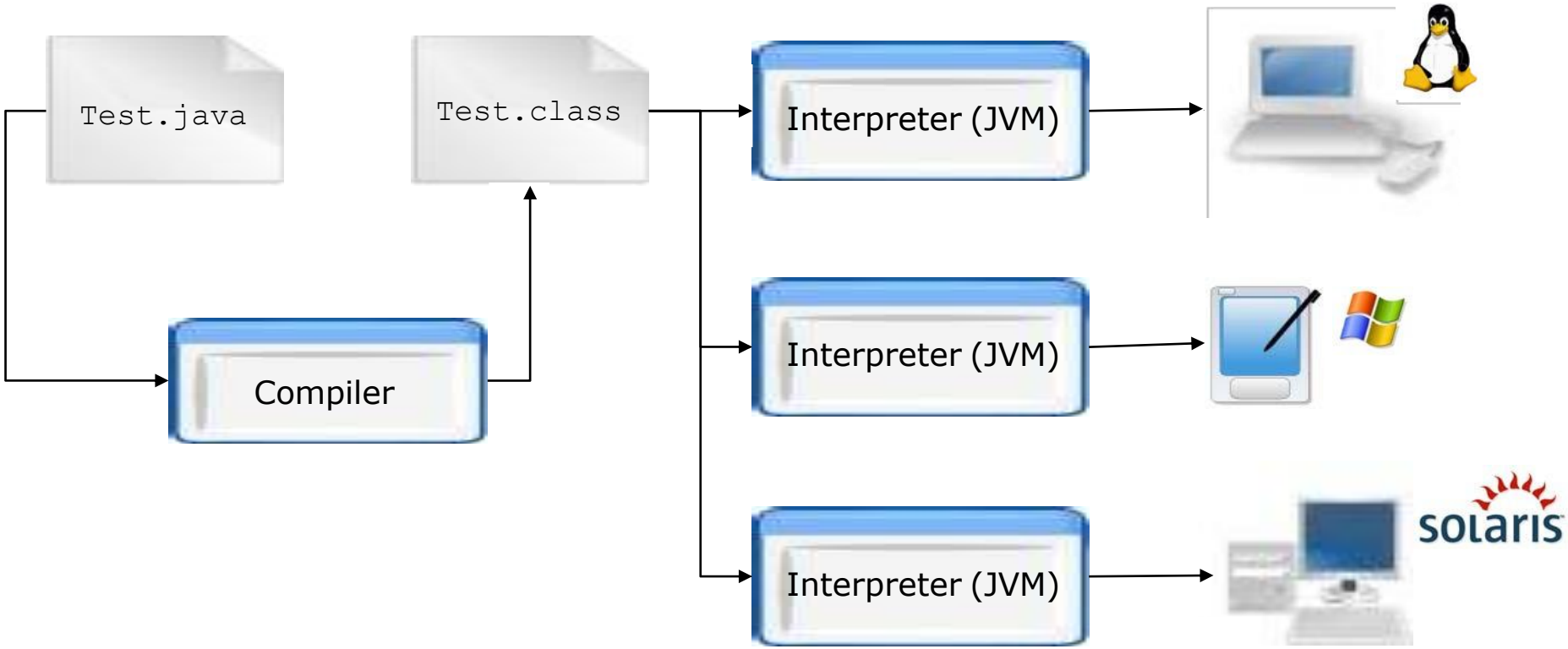
— A Java applet is a program designed to be executed under a Java capable browser.

The Java platform

Java programs are compiled to Java byte-codes, a kind of machine independent representation.
The program is then executed by an interpreter called the Java Virtual Machine (JVM).



The compiled code is independent of the architecture of the computer.
The price to pay is a slower execution.



EXAMPLE

```java
/**
*  Hello World Application
*  Our first example
*/
public class HelloWorld {
   public static void main(String[] args) { System.out.println("Hello World!"); //
      display output
   }
}
```

```
$ javac HelloWorld.java

$ ls
HelloWorld.class
HelloWorld.java

$ java HelloWorld
Hello World
```

Deutsche Bank
Corporate Division

DOCUMENTATION

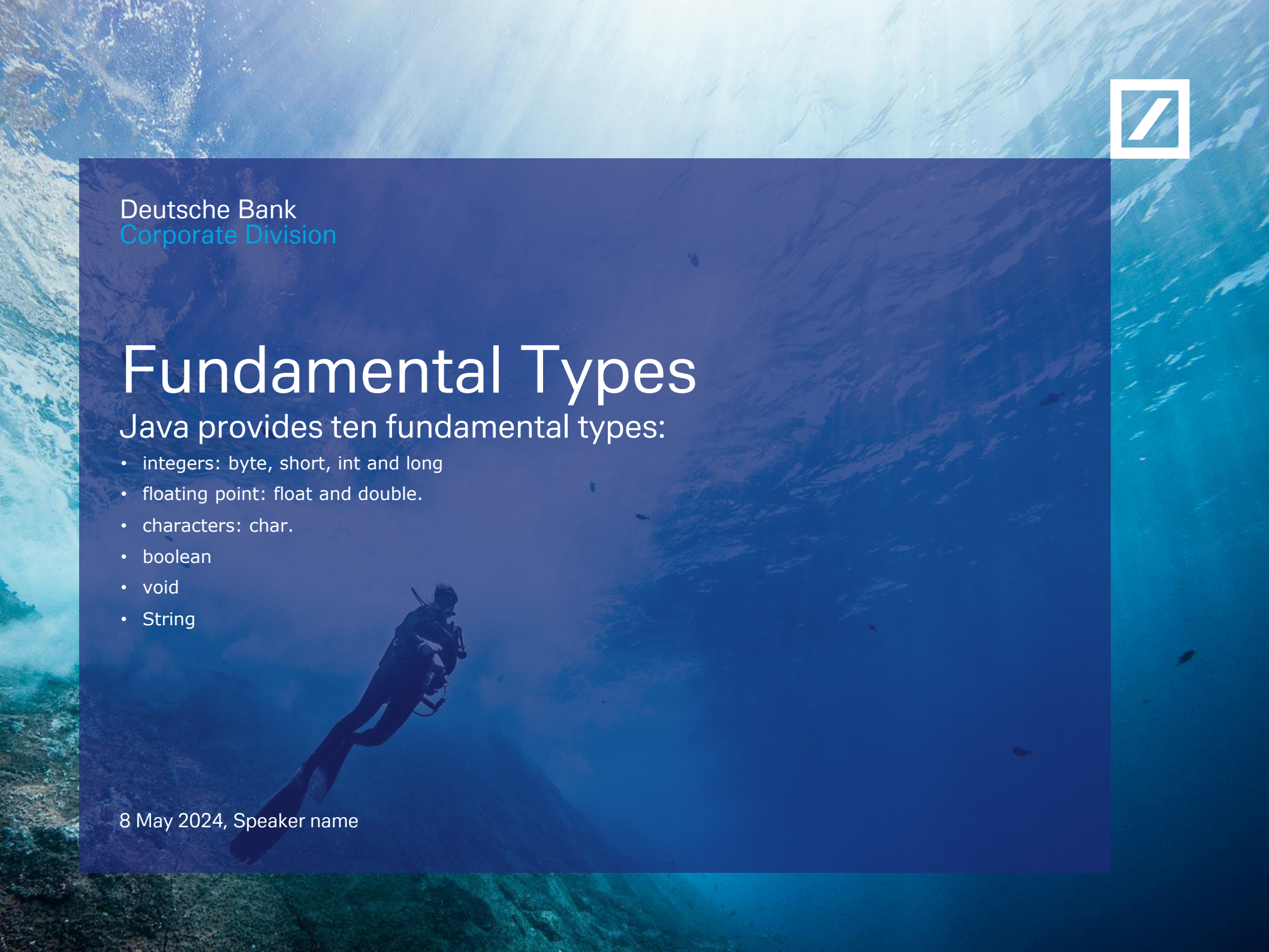
The javadoc utility can be used to generate automatically documentation for the class.

```
/**
* My first <b>Test</b>
* @author Xyz
* @version 1.1
*/
public class HelloWorld {
   /**
   * @param args the command line arguments
   * @since 1.0
   */
   public static void main(String[] args) {
      System.out.println("Hello World");
   }
}
```

Deutsche Bank
Corporate Division

Exploring Fundamental Data Types, Variables, and Operators

- Ten Fundamental Types
- Variables
- Literals
- Constants
- Expressions
- Arithmetic Expressions
- Relational Expressions
- Bit Level Operators
- Logical Operators



Deutsche Bank
Corporate Division

Fundamental Types

Java provides ten fundamental types:

- integers: byte, short, int and long
- floating point: float and double.
- characters: char.
- boolean
- void
- String

8 May 2024, Speaker name

VARIABLES

The variables are declared specifying its type and name, and initialized in the point of declaration, or later with the assignment expression:

```
int x;
double f = 0.33;
char c = 'a';
String s = "abcd";

x = 55;
```

LITERALS

The integer values can be written in decimal, hexadecimal, octal and long forms:

```
int x = 34;             // decimal value
int y = 0x3ef;          // hexadecimal
int z = 0772;           // octal
long m = 240395922L; // long
```

The floating point values are of type double by default:

```
double d = 6.28;        // 6.28 is a double value
float f = 6.28F;        // 6.28F is a float value
```

LITERALS

The character values are specified with the standard C notation, with extensions for Unicode values:

```
char c = 'a';          // character lowercase a
char d = '\n';         // newline
char e = '\u2122'      // unicode character (TM)
```

The boolean values are true and false:

```
boolean ready = true; // boolean value true
boolean late = false; // boolean value false
```

CONSTANTS

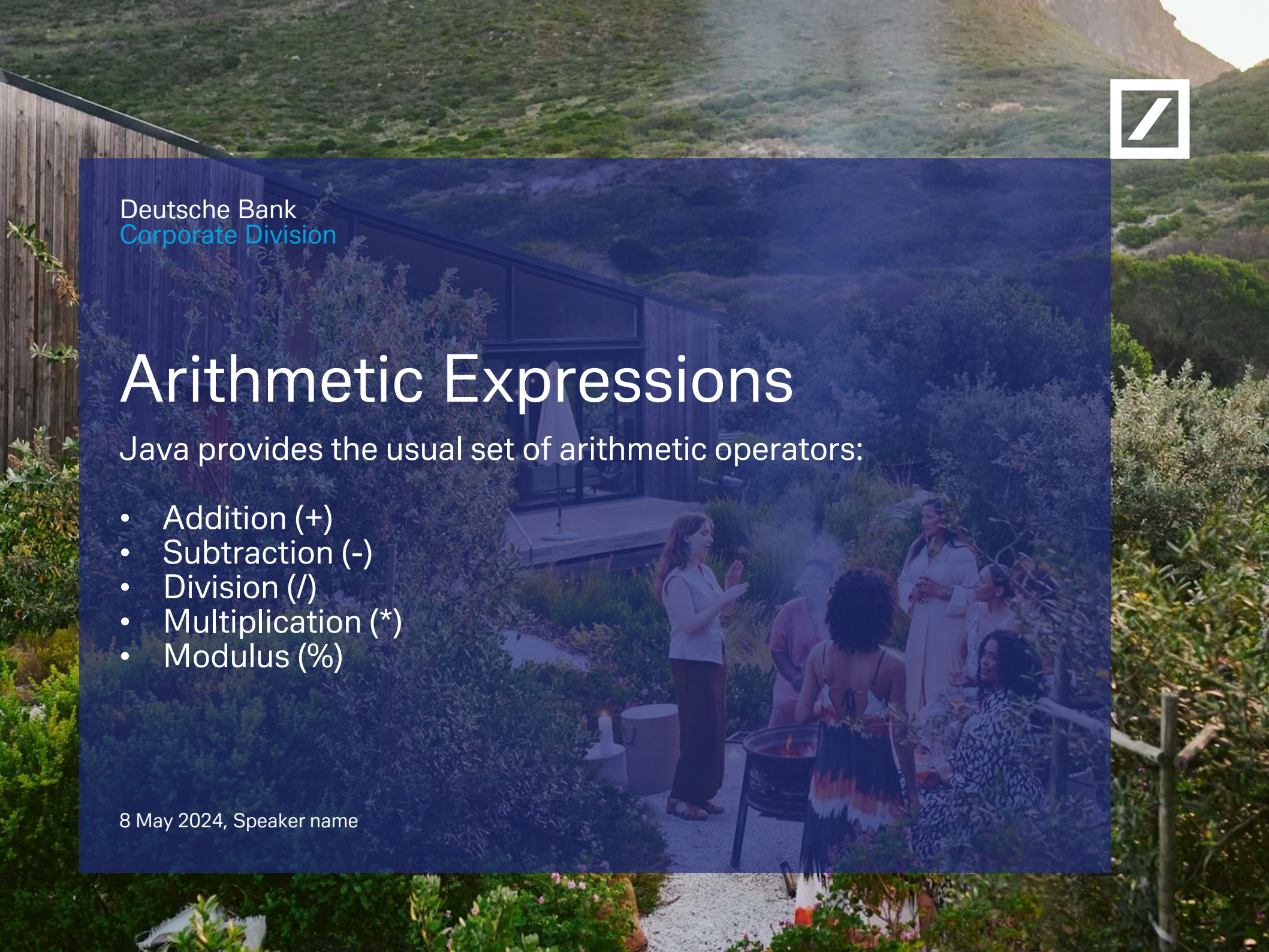
Constants are declared with the word final in front. The specification of the initial value is compulsory:

```
final double pi = 3.1415;      // constant PI
final int maxSize = 100;       // integer constant
final char lastLetter = 'z';   // last lowercase letter
final String word = "Hello";   // a constant string
```

EXPRESSIONS

Java provides a rich set of expressions:

- Arithmetic
- Bit level
- Relational
- Logical
- Strings related



Deutsche Bank
Corporate Division

Arithmetic Expressions

Java provides the usual set of arithmetic operators:

- Addition (+)
- Subtraction (-)
- Division (/)
- Multiplication (*)
- Modulus (%)

8 May 2024, Speaker name

ARITHMETIC OPERATORS

```
class Arithmetic {
  public static void main(String[] args) {
    int x = 12;
    int y = 2 * x;
    System.out.println(y);
    int z = (y - x) % 5;
    System.out.println(z);
    final float pi = 3.1415F;
    float f = pi / 0.62F;
    System.out.println(f);
  }
}
```

```
$ java Arithmetic
24
2
5.0669355
```

ARITHMETIC OPERATORS

Shorthand operators are provided:

```
class ShortHand {
  public static void main(String[] args) {
    int x = 12;

    x += 5;                          // x = x + 5
    System.out.println(x);

    x *= 2;                          // x = x * 2
    System.out.println(x);
  }
}
```

```
$ java ShortHand
17
34
```

ARITHMETIC OPERATORS

Pre and post operators are also provided:

```
  class Increment {
    public static void
      main(String[] args) { int x =
      12,y = 12;


    System.out.println(x++); // printed
    and then incremented
    System.out.println(x);

    System.out.println(++y); //
    incremented and then printed
    System.out.println(y);
  }
}
```

```
$ java Increment
12 13 13 13
```

Deutsche Bank
Corporate Division

RELATIONAL EXPRESSIONS

Java provides the following relational operators:

- Equivalent (==)
- Not equivalent (!=)
- Less than (<)
- Greater that (>)
- Less than or equal (<=)
- Greater than or equal (>=)

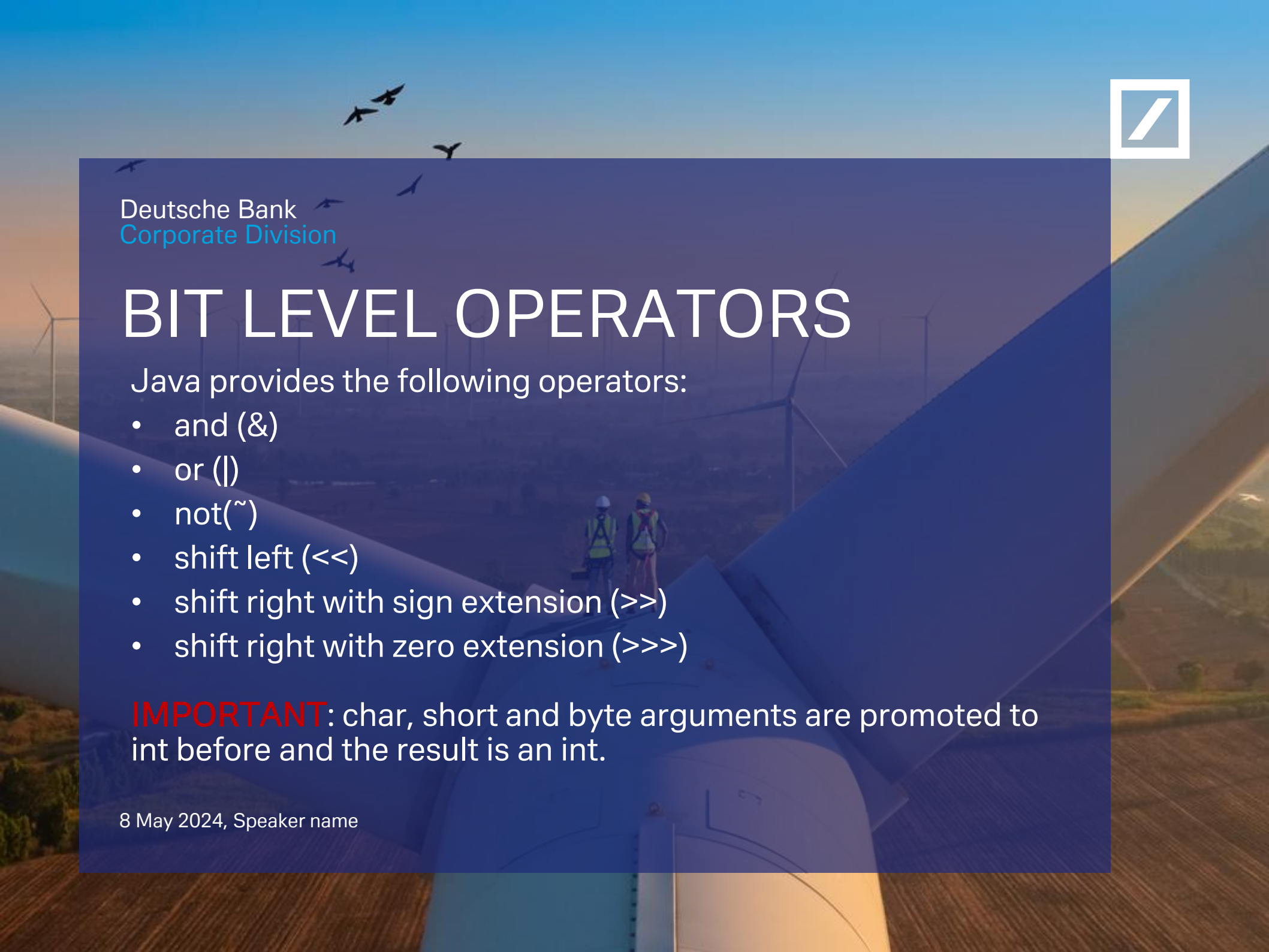
IMPORTANT: Relational expressions always return a boolean value.

8 May 2024, Speaker name

RELATIONAL EXPRESSIONS

```
class Boolean {
  public static void main(String[] args) {
    int x = 12,y = 33;

    System.out.println(x < y);
    System.out.println(x != y - 21);

    boolean test = x >= 10;
    System.out.println(test);
  }
}
```

```
$ java Boolean
true
false
true
```



Deutsche Bank
Corporate Division

BIT LEVEL OPERATORS

Java provides the following operators:

- and (&)
- or (|)
- not(~)
- shift left (<<)
- shift right with sign extension (>>)
- shift right with zero extension (>>>)

IMPORTANT: char, short and byte arguments are promoted to int before and the result is an int.

8 May 2024, Speaker name

BIT LEVEL OPERATORS

```
class Bits {
  public static void main(String[] args) {

    int x = 0x16;                  // 00000000000000000000000000010110
    int y = 0x33;                  // 00000000000000000000000000110011

    System.out.println(x & y);// 00000000000000000000000000010010
    System.out.println(x | y);// 00000000000000000000000000110111
    System.out.println(~x);    // 11111111111111111111111111101001

    x &= 0xf;                      // 00000000000000000000000000000110
    System.out.println(x);         // 00000000000000000000000000000110

    short s = 7;                   // 0000000000000111
    System.out.println(~s);        // 11111111111111111111111111111000
  }
}
```

BIT LEVEL OPERATORS

```
class Bits2 {
  public static void main(String[] args) {
    int x = 0x16;                    //00000000000000000000000000010110
    System.out.println(x << 3);//00000000000000000000000010110000

    int y = 0xfe;                    //00000000000000000000000011111110
    y >>= 4;                         //00000000000000000000000000001111
    System.out.println(y);           //00000000000000000000000000001111

    x = 9;                           //00000000000000000000000000001001
    System.out.println(x >> 3);//00000000000000000000000000000001
    System.out.println(x >>>3);//00000000000000000000000000000001

    x = -9;                          //11111111111111111111111111110111
    System.out.println(x >> 3);//11111111111111111111111111111110
    System.out.println(x >>>3);//00011111111111111111111111111110
  }
}
```

Deutsche Bank
Corporate Division

LOGICAL OPERATORS

Java provides the following operators:

- and (&&)
- or (||)
- not(!)

IMPORTANT: The logical operators can only be applied to boolean expressions and return a boolean value.

8 May 2024, Speaker name

LOGICAL OPERATORS

```
class Logical {
  public static void main(String[] args) {
    int x = 12,y = 33;
    double d = 2.45,e = 4.54;

    System.out.println(x < y && d < e);
    System.out.println(!(x < y));

    boolean test = 'a' > 'z';
    System.out.println(test || d - 2.1 > 0);
  }
}
```

```
$ java Logical
true
false
true
```

Deutsche Bank
Corporate Division

CONTROL STRUCTURES

Java provides the same set of control structures than C.

IMPORTANT: the value used in the conditional expressions must be a boolean.

8 May 2024, Speaker name

CONTROL STRUCTURES (if)

```
class If {
  public static void main(String[] args) {
    char c = 'x';

    if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
      System.out.println("letter: " + c);
    else
      if (c >= '0' && c <= '9')
        System.out.println("digit: " + c);
      else {
        System.out.println("the character is: " + c);
        System.out.println("it is not a letter");
        System.out.println("and it is not a digit");
      }
  }
}
```

```
$ java If
letter: x
```

CONTROL STRUCTURES (while)

```
class While {
  public static void main(String[] args) {
    final float initialValue = 2.34F;
    final float step = 0.11F;
    final float limit = 4.69F;
    float var = initialValue;

    int counter = 0;
    while (var < limit) {
      var += step;
      counter++;
    }
    System.out.println("Incremented " + counter + " times");
  }
}
```

```
$ java While
Incremented 22 times
```

CONTROL STRUCTURES (for)

```
class For {
  public static void main(String[] args) {
    final float initialValue = 2.34F;
    final float step = 0.11F;
    final float limit = 4.69F;
    int counter = 0;

    for (float var = initialValue;var < limit;var += step)
      counter++;
    System.out.println("Incremented " + counter + " times");
  }
}
```

```
$ java For
Incremented 22 times
```

CONTROL STRUCTURES (break/continue)

```
class BreakContinue {
  public static void main(String[] args) {

    for (int counter = 0;counter < 10;counter++) {

      // start a new iteration if the counter is odd
      if (counter % 2 == 1) continue;

      // abandon the loop if the counter is equal to 8
      if (counter == 8) break;

      // print the value
      System.out.println(counter);
    }
    System.out.println("done.");
  }
}
```

```
$ java BreakContinue
0 2 4 6 done.
```

CONTROL STRUCTURES (switch)

```
class Switch {
  public static void main(String[] args) {

    boolean leapYear = true;
    int days = 0;

    for(int month = 1;month <= 12;month++) {
      switch(month) {
        case 1: // months with 31 days
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
          days += 31;
          break;
```

CONTROL STRUCTURES (switch)

```
        case 2: // February is a special case
          if (leapYear)
            days += 29;
          else
            days += 28;
          break;
        default: // it must be a month with 30 days
          days += 30;
          break;
        }
      }
      System.out.println("number of days: " + days);
    }
  }
```

```
$ java Switch
number of days: 366
```

Deutsche Bank
Corporate Division

Arrays

- All about Arrays
- Components of Arrays
- Common Line Arguments

ARRAYS

Arrays can be used to store a number of elements of the same type:

```
int[] a;          // an unitialized array of integers
float[] b;        // an unitialized array of floats
String[] c;       // an unitialized array of Strings
```

IMPORTANT: The declaration does not specify a size. However, it can be inferred when initialized:

```
int[] a = {13,56,2034,4,55};              // size: 5
float[] b = {1.23F,2.1F};                 // size: 2
String[] c = {"Java","is","great"};       // size: 3
```

ARRAYS

Other possibility to allocate space for arrays consists in the use of the operator new:

```
int i = 3,j = 5;
double[] d;                 // unitialized array of doubles

d = new double[i+j];        // array of 8 doubles
```

Components of the arrays are initialized with default values:

- 0 for numeric type elements,
- '\0' for characters
- null for references.

ARRAYS

Components can be accessed with an integer index with values from 0 to length minus 1.

```
a[2] = 1000; // modify the third element of a
```

Every array has a member called length that can be used to get the length of the array:

```
int len = a.length; // get the size of the array
```

ARRAYS

```
class Arrays {
  public static void main(String[] args) {
    int[] a = {2,4,3,1};

    // compute the summation of the elements of a
    int sum = 0;
    for(int i = 0;i < a.length;i++) sum += a[i];

    // create an array of the size computed before
    float[] d = new float[sum];
    for(int i = 0;i < d.length;i++) d[i] = 1.0F / (i+1);

     // print values in odd positions
    for(int i = 1;i < d.length;i += 2)
      System.out.println("d[" + i + "]=" + d[i]);
  }
}
```

```
$ java Arrays
d[1]=0.5 d[3]=0.25 d[5]=0.16666667 d[7]=0.125 d[9]=0.1
```

COMMAND LINE ARGUMENTS

We have seen that the method main has to be defined as follows:

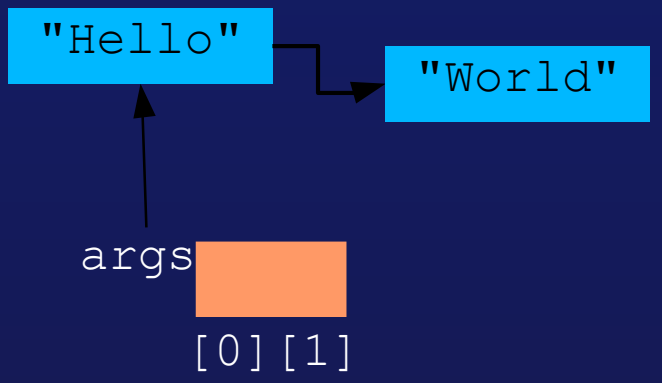
```
public static void main(String[] args)
```

Through the array argument, the program can get access to the command line arguments

COMMON LINE ARGUMENTS

```
class CommandArguments {
  public static void main(String[] args) {
    for(int i = 0;i < args.length;i++)
      System.out.println(args[i]);
  }
}
```

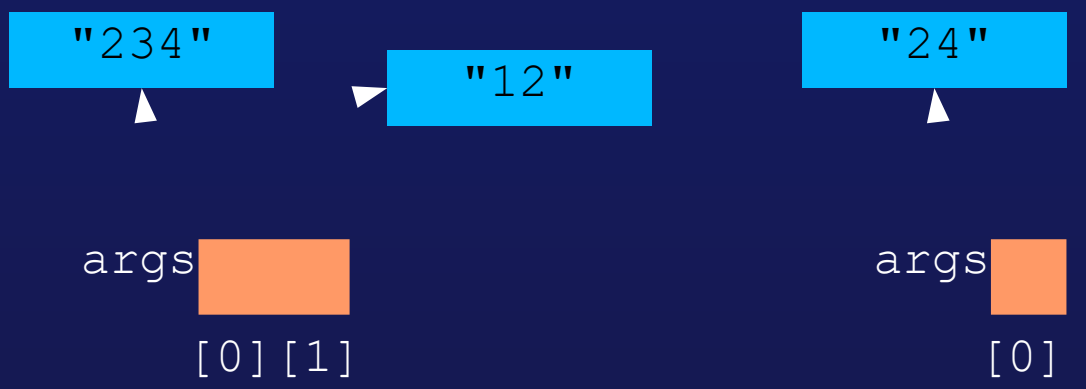
```
$ java CommandArguments Hello World
Hello
World

$ java CommandArguments

$ java CommandArguments I have 25 cents
I
have
25
cents
```

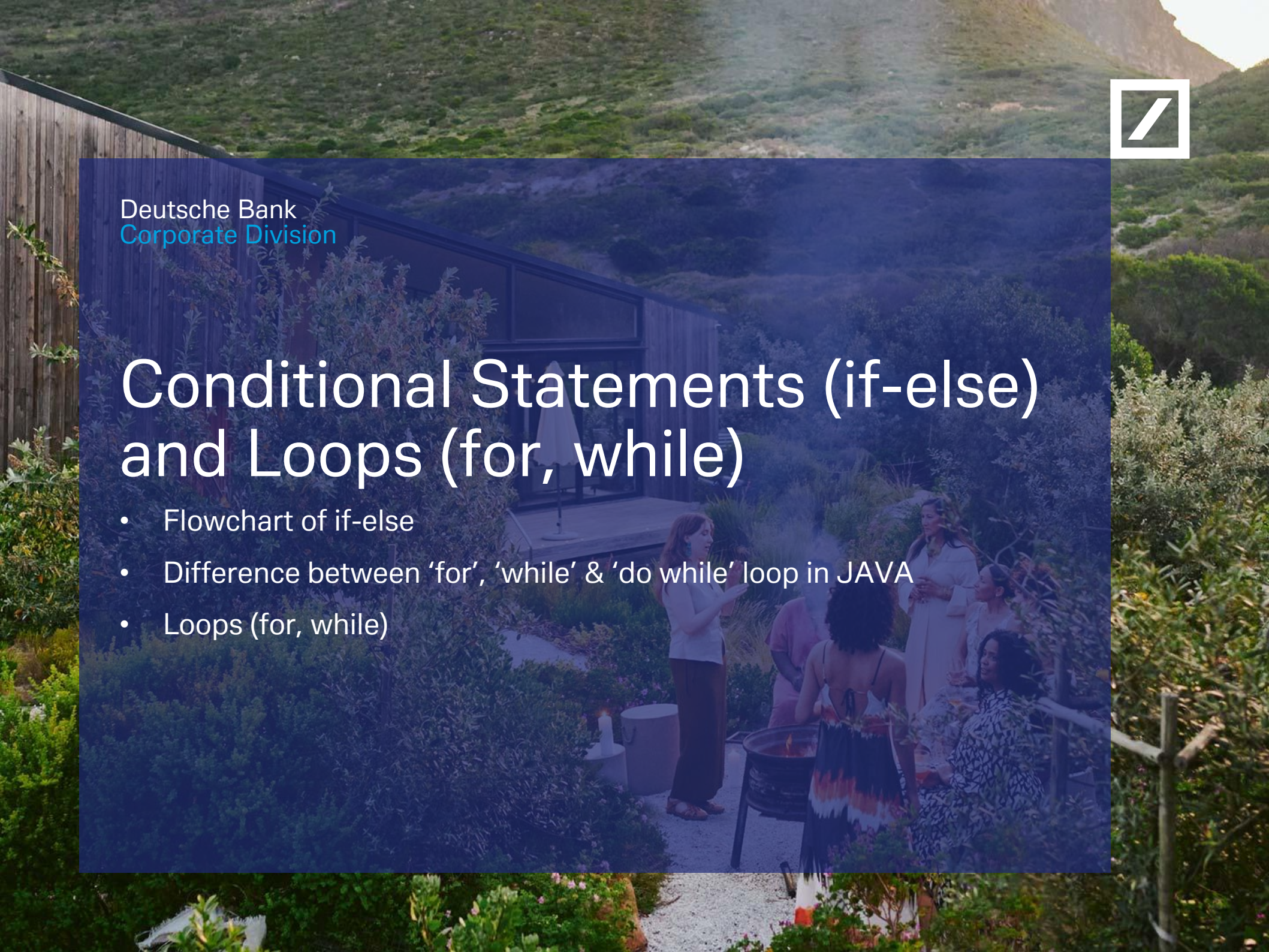


COMMON LINE ARGUMENTS

```
class Add {
  public static void main(String[] args) {
    if (args.length != 2) {
      System.out.println("Error");
      System.exit(0);
    }
    int arg1 = Integer.parseInt(args[0]);
    int arg2 = Integer.parseInt(args[1]);
    System.out.println(arg1 + arg2);
  }
}
```

```
$ java Add 234 12
246

$ java Add 24
Error
```





Deutsche Bank
Corporate Division

Conditional Statements (if-else) and Loops (for, while)

- Flowchart of if-else
- Difference between ‘for’, ‘while’ & ‘do while’ loop in JAVA
- Loops (for, while)

IF - ELSE

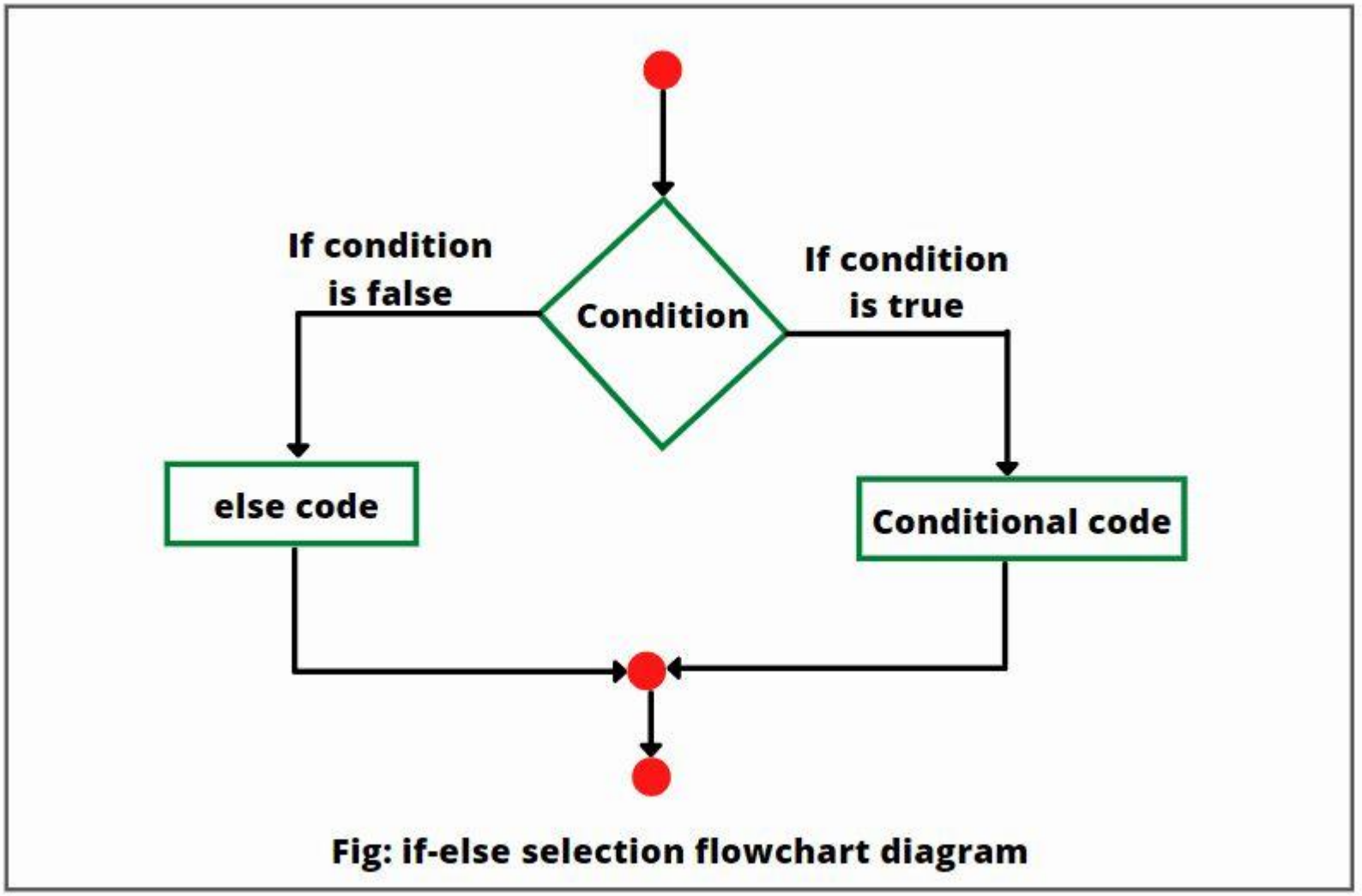


Fig: if-else selection flowchart diagram

LOOPS (for, while)

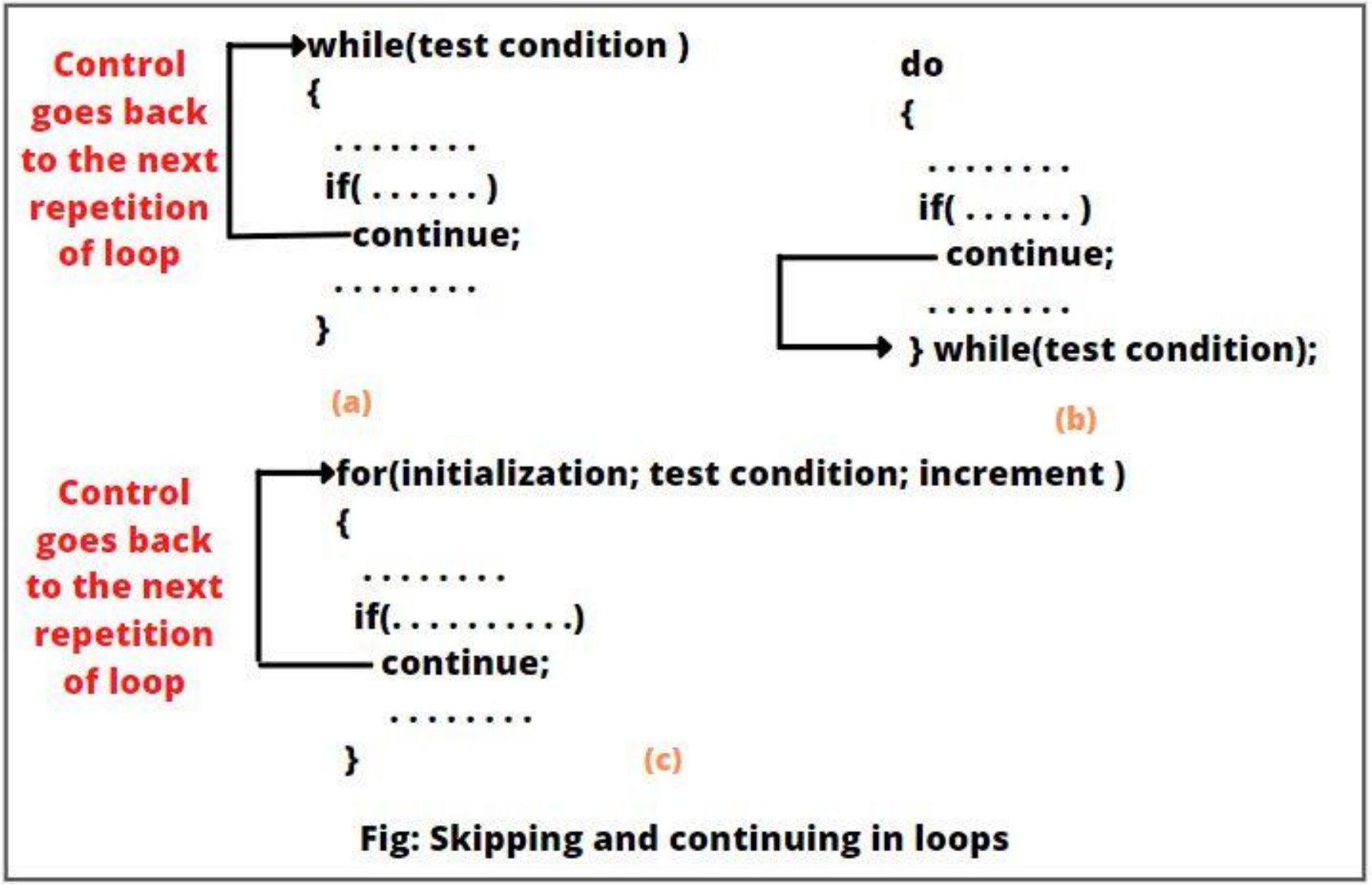


Fig: Skipping and continuing in loops

Difference between ‘for’, ‘while’ and ‘do while’ loop in JAVA

Aspect	for Loop	while Loop	do-while Loop
Usage	Known number of iterations	Unknown number of iterations	At least one execution needed
Execution	Repeats as long as the condition is true	Repeats as long as the condition is true	Executes once, then repeats if condition is true
Initiation	Initialized in the loop statement	Initialized before the loop	Initialized before the loop
Condition	Checked before each iteration	Checked before each iteration	Checked after each iteration
Update	Updated in the loop statement	Updated within the loop body	Updated within the loop body

Deutsche Bank
Corporate Division

CLASSES

- All About Classes
- Class Variables
- Class Methods
- Methods
- Equality and Equivalence
- Instance Initiation

CLASSES

A class is defined in Java by using the class keyword and specifying a name for it:

```
class Book {
}
```

New instances of the class can be created with new:

```
Book b1 = new Book();
Book b2 = new Book();

b3 = new Book();
```

CLASSES

Inside a class it is possible to define:

- data elements, usually called instance variables
- functions, usually called methods

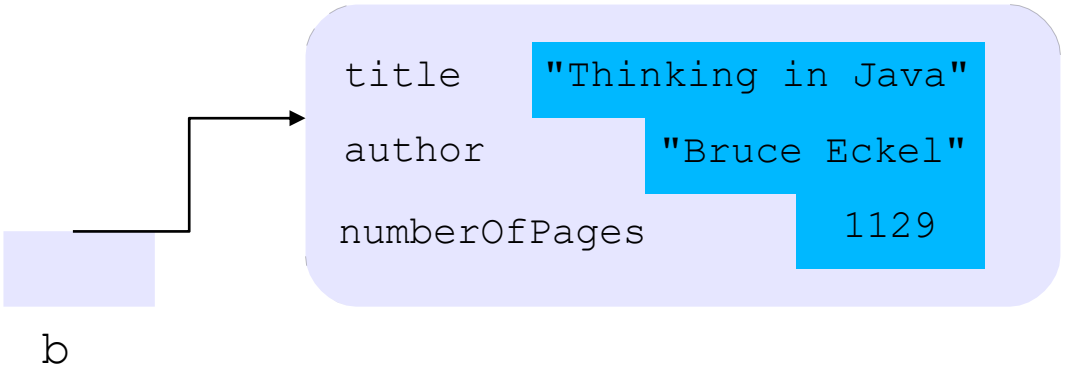
Class Book with instance variables:

```
class Book {
  String title;
  String author;
  int numberOfPages;
}
```

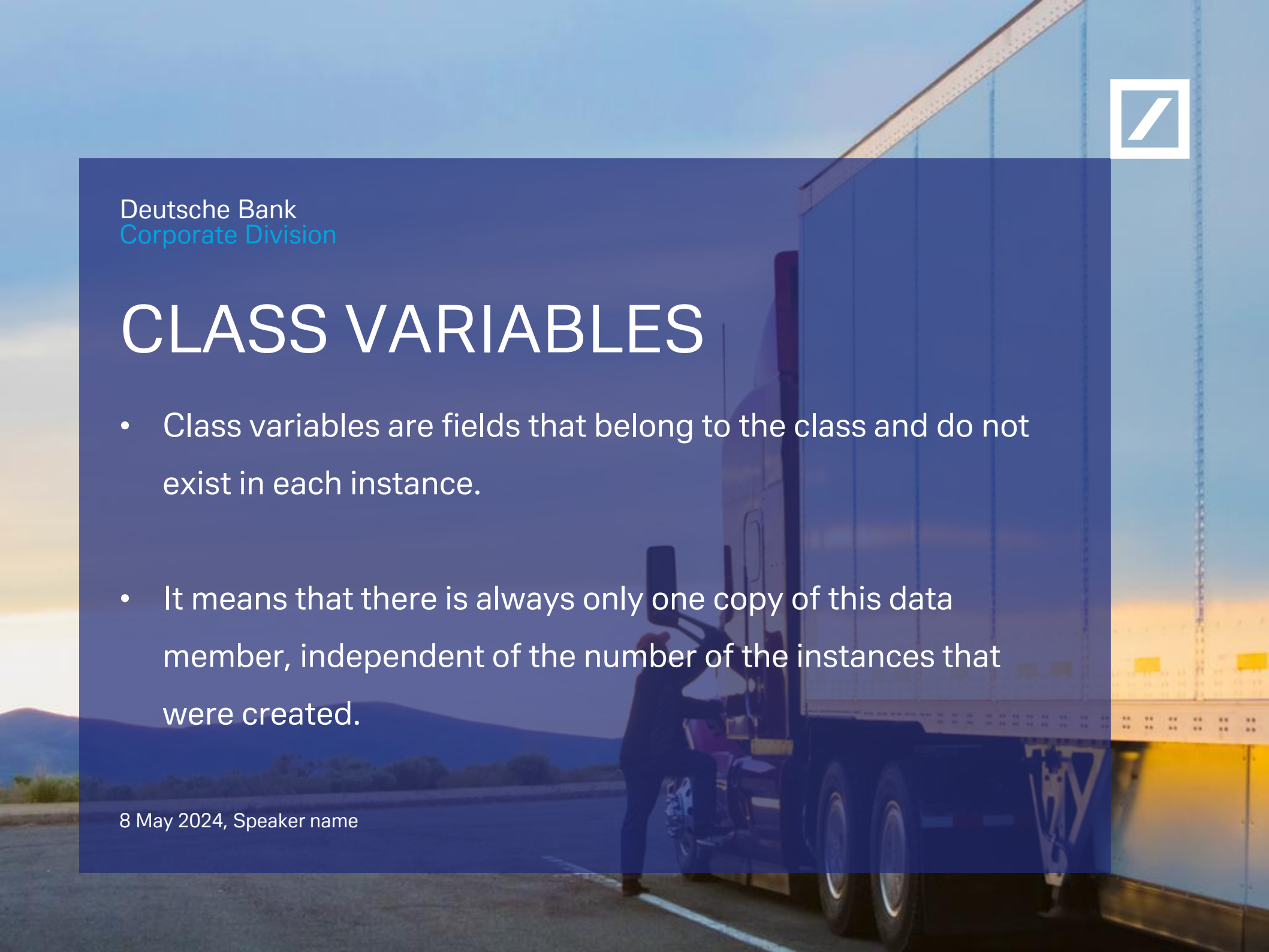
The instance variables can be accessed with the dot notation.

CLASSES

```
class Book {
  String title;
  String author;
  int numberOfPages;
}
```




```
class ExampleBooks {
  public static void main(String[] args) {

    Book b = new Book();
    b.title = "Thinking in Java";
    b.author = "Bruce Eckel";
    b.numberOfPages = 1129;
    System.out.println(b.title + " : " + b.author +
                       " : " + b.numberOfPages);
  }
}
```



Deutsche Bank
Corporate Division

CLASS VARIABLES

- Class variables are fields that belong to the class and do not exist in each instance.

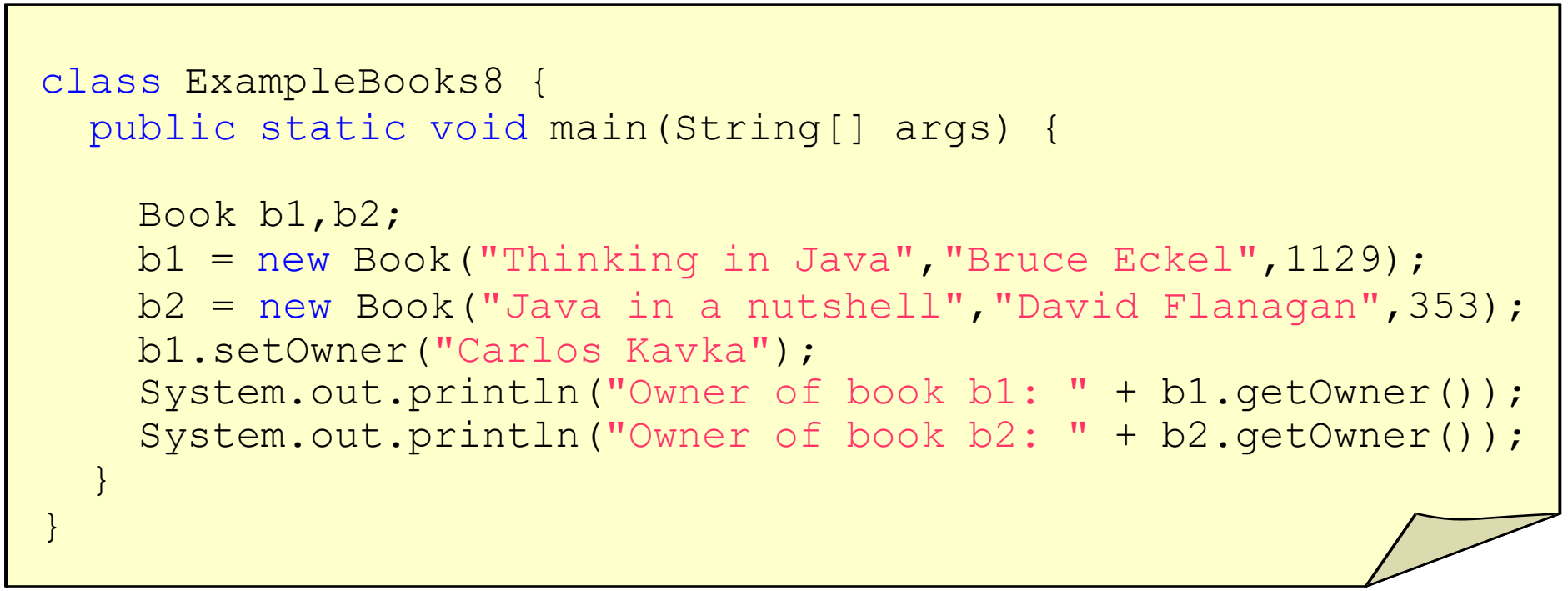
- It means that there is always only one copy of this data member, independent of the number of the instances that were created.

8 May 2024, Speaker name

CLASS VARIABLES

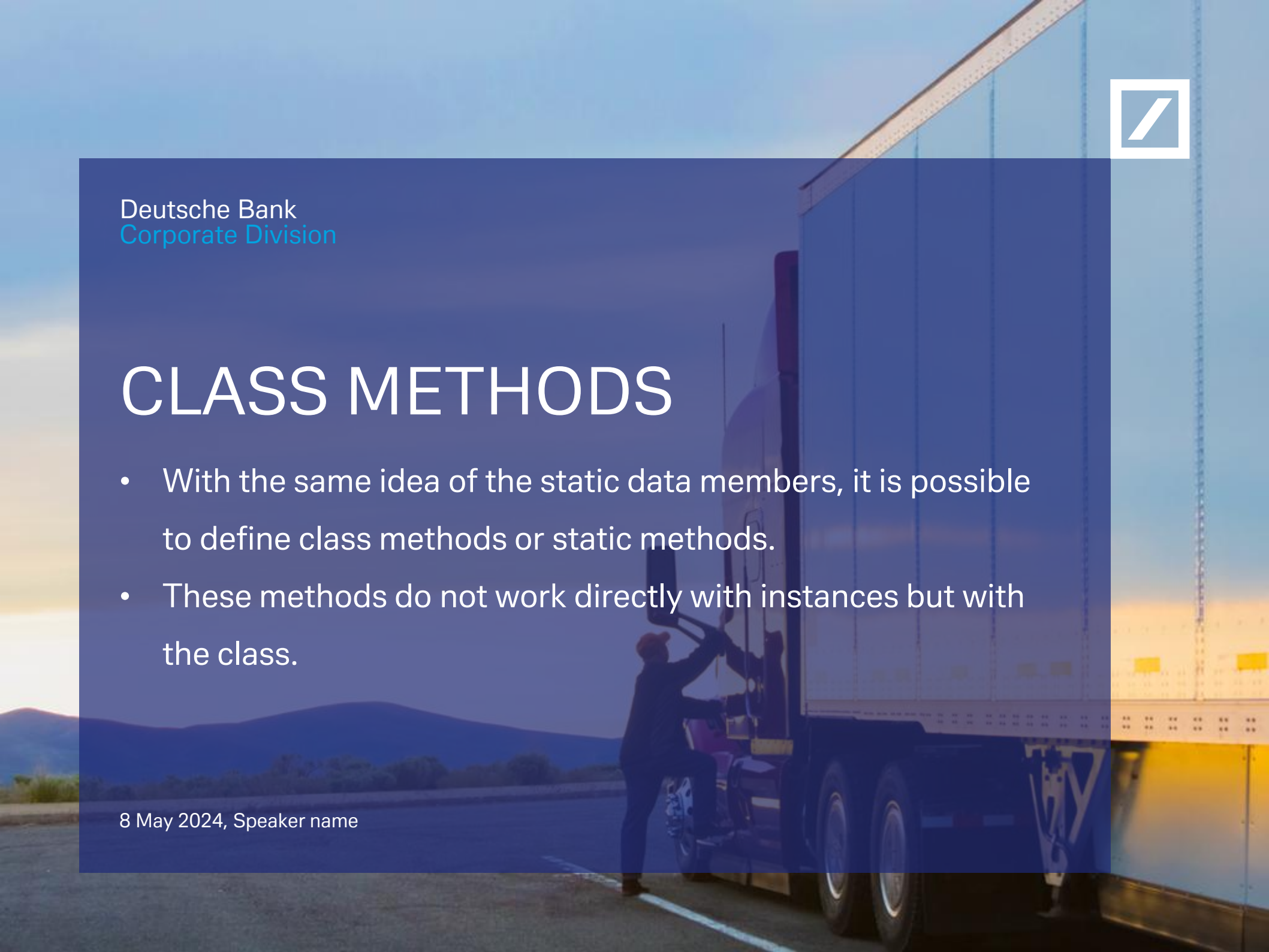
```
class Book {
  String title;
  String author;
  int numberOfPages;
  String ISBN;
  static String owner;

  ...

  public void setOwner(String name) {
    owner = name;
  }
  public String getOwner() {
    return owner;
  }
}
```

CLASS VARIABLES



```
class ExampleBooks8 {
  public static void main(String[] args) {

    Book b1,b2;
    b1 = new Book("Thinking in Java","Bruce Eckel",1129);
    b2 = new Book("Java in a nutshell","David Flanagan",353);
    b1.setOwner("Carlos Kavka");
    System.out.println("Owner of book b1: " + b1.getOwner());
    System.out.println("Owner of book b2: " + b2.getOwner());
  }
}
```

```
$ java ExampleBooks8
Owner of book b1: Carlos Kavka
Owner of book b2: Carlos Kavka
```



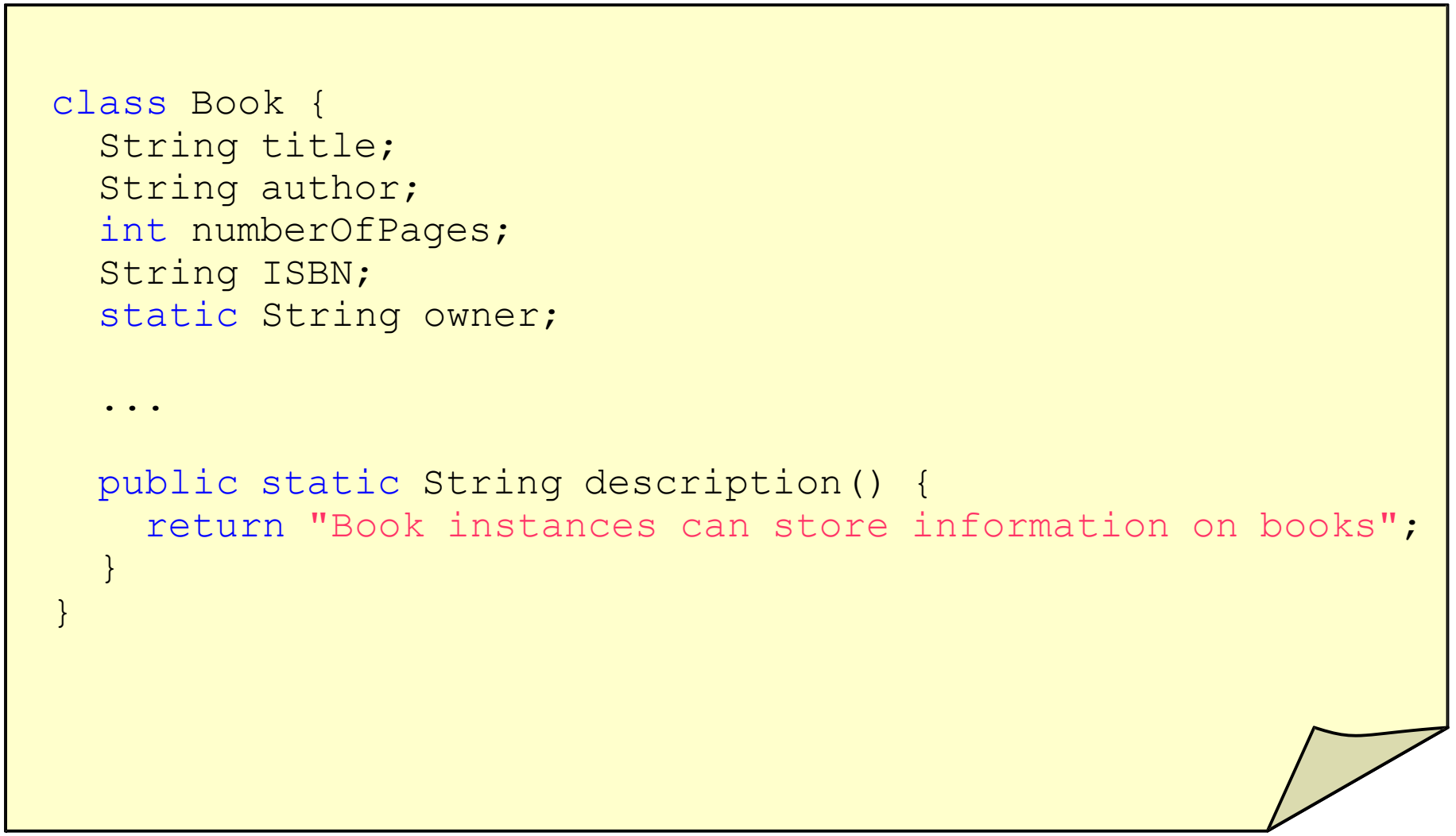
Deutsche Bank
Corporate Division

CLASS METHODS

- With the same idea of the static data members, it is possible to define class methods or static methods.
- These methods do not work directly with instances but with the class.

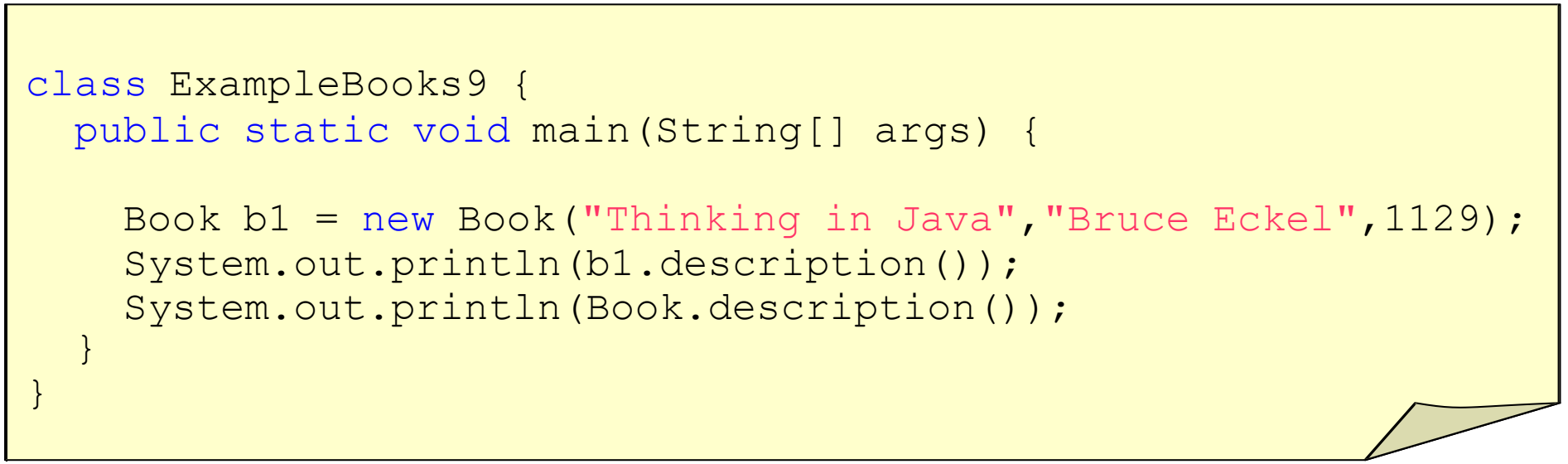
8 May 2024, Speaker name

CLASS METHODS



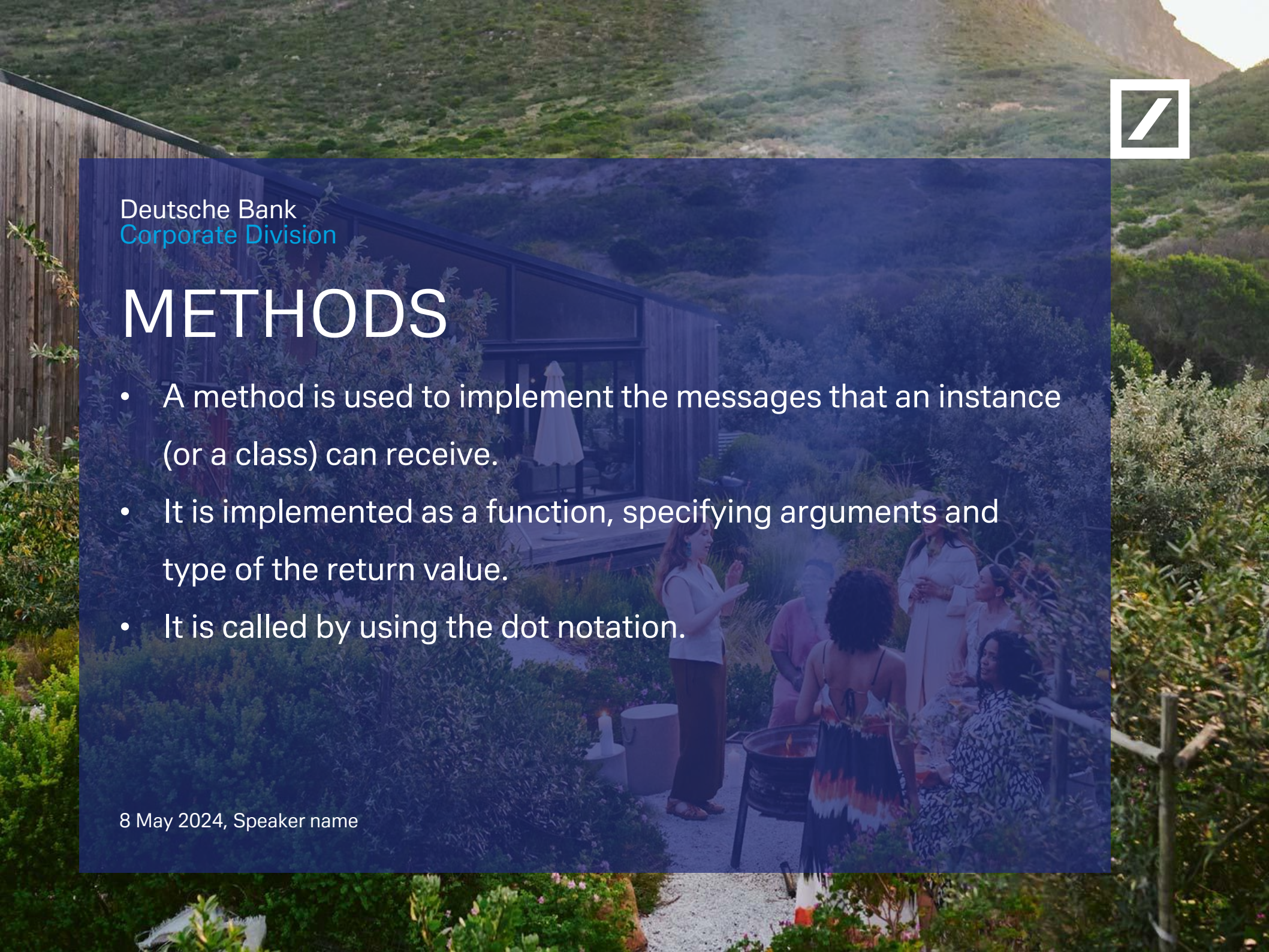
```
class Book {
  String title;
  String author;
  int numberOfPages;
  String ISBN;
  static String owner;

  ...

  public static String description() {
    return "Book instances can store information on books";
  }
}
```

CLASS METHODS



```
class ExampleBooks9 {
  public static void main(String[] args) {

    Book b1 = new Book("Thinking in Java","Bruce Eckel",1129);
    System.out.println(b1.description());
    System.out.println(Book.description());
  }
}
```

```
$ java ExampleBooks9
Book instances can store information on books
Book instances can store information on books
```



Deutsche Bank
Corporate Division

METHODS

- A method is used to implement the messages that an instance (or a class) can receive.
- It is implemented as a function, specifying arguments and type of the return value.
- It is called by using the dot notation.

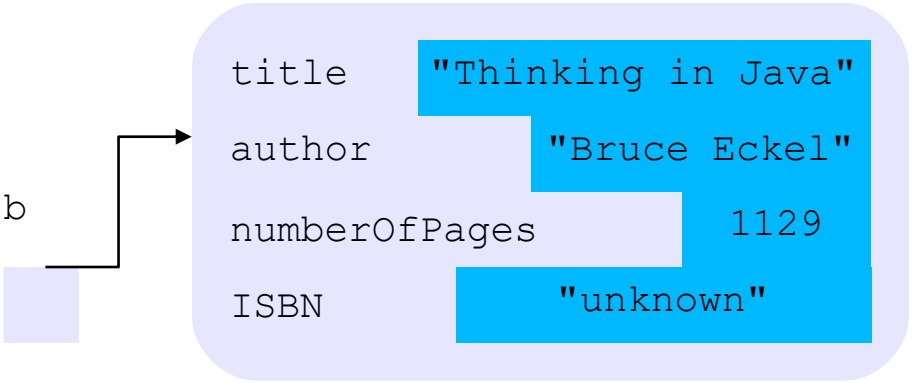
8 May 2024, Speaker name

METHODS

```
class Book {
  String title;
  String author;
  int numberOfPages;
  String ISBN;

  ...

  // compute initials of author's name
  public String getInitials() {
    String initials = "";
    for(int i = 0;i < author.length();i++) {
      char currentChar = author.charAt(i);
      if (currentChar >= 'A' && currentChar <='Z')
        initials = initials + currentChar + '.';
    }
    return initials;
  }
}
```

```
class ExampleBooks4 {
  public static void main(String[] args) {
    Book b;

    b = new Book("Thinking in Java","Bruce Eckel",1129);
    System.out.println("Initials: " + b.getInitials());
  }
}
```

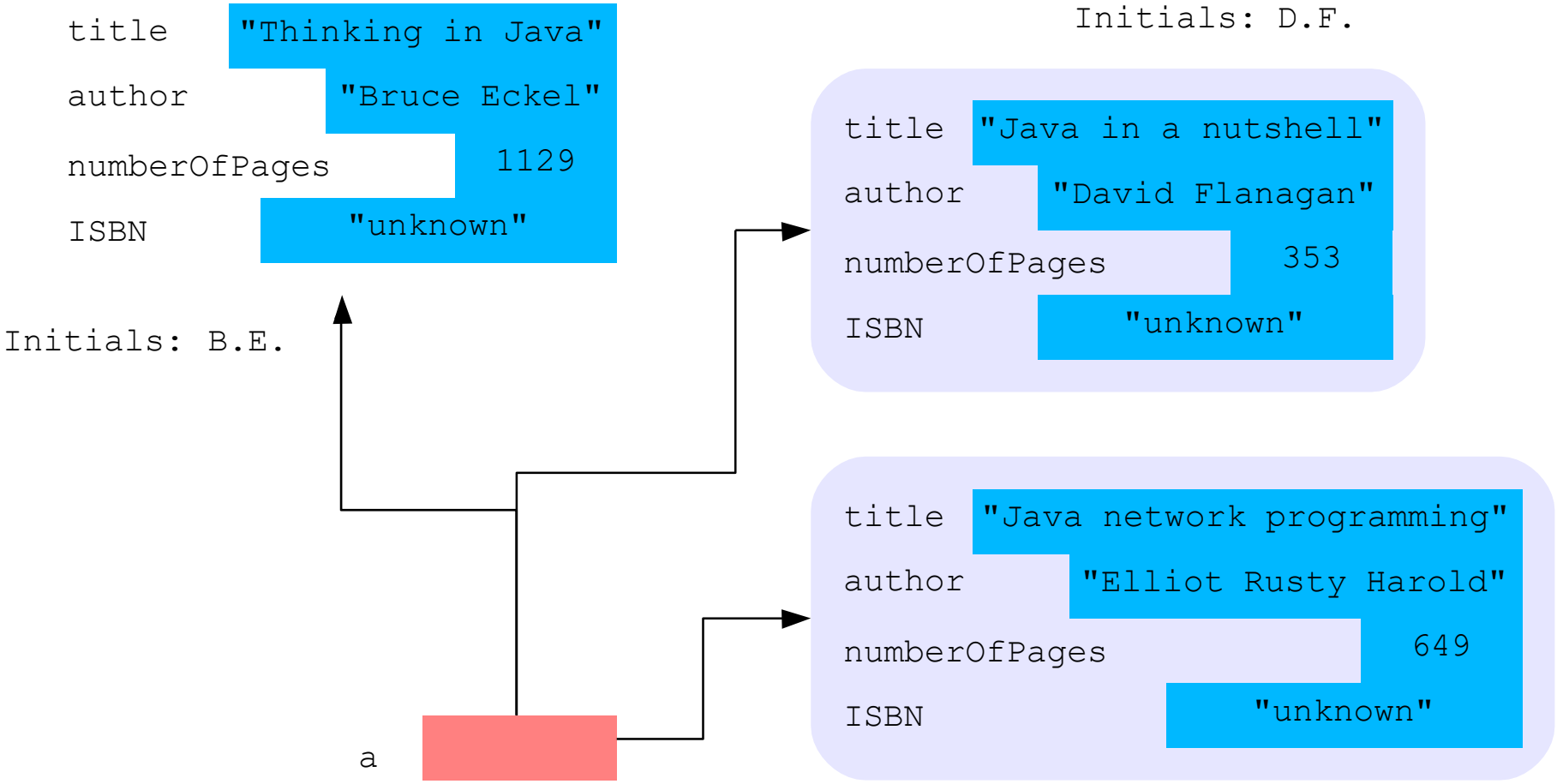
```
$ java ExampleBooks4
Initials: B.E.
```



```
class ExampleBooks5 {
  public static void main(String[] args) {

    Book[] a = new Book[3];
    a[0] = new Book("Thinking in Java","Bruce Eckel",1129);
    a[1] = new Book("Java in a nutshell","David Flanagan",353);
    a[2] = new Book("Java network programming",
                    "Elliott Rusty Harold",649);

    for(int i = 0;i < a.length;i++)
      System.out.println("Initials: " + a[i].getInitials());
  }
}
```

```
$ java ExampleBooks5
Initials: B.E.
Initials: D.F.
Initials: E.R.H.
```

METHODS

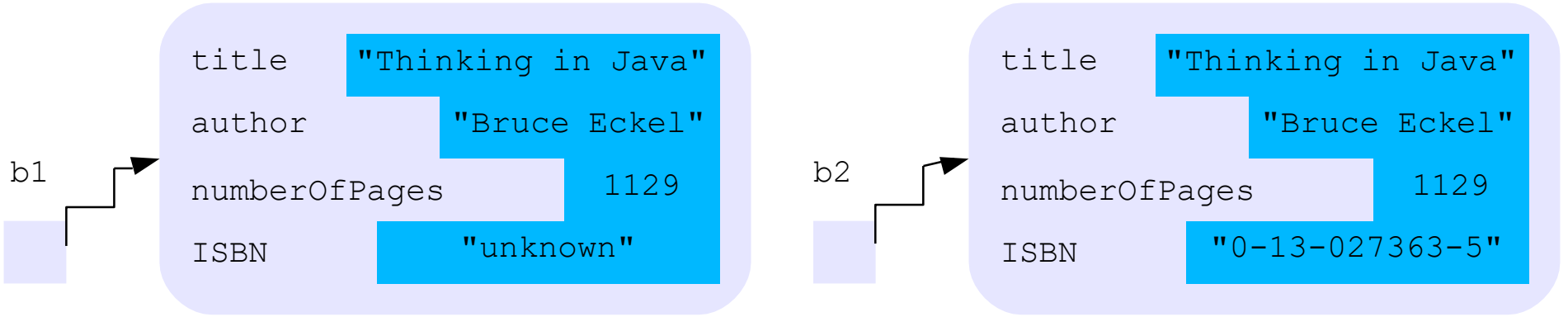

title "Thinking in Java"
author "Bruce Eckel"
numberOfPages 1129
ISBN "unknown"
Initials: B.E.
Initials: D.F.
title "Java in a nutshell"
author "David Flanagan"
numberOfPages 353
ISBN "unknown"
title "Java network programming"
author "Elliot Rusty Harold"
numberOfPages 649
ISBN "unknown"
a

EQUALITY AND EQUIVALENCE

```
class ExampleBooks6 {
  public static void main(String[] args) {

    Book b1,b2;

    b1 = new Book("Thinking in Java","Bruce Eckel",1129);
    b2 = new Book("Thinking in Java","Bruce Eckel",1129);

    if (b1 == b2)
      System.out.println("The two books are the same");
    else
      System.out.println("The two books are different");
  }
}
```

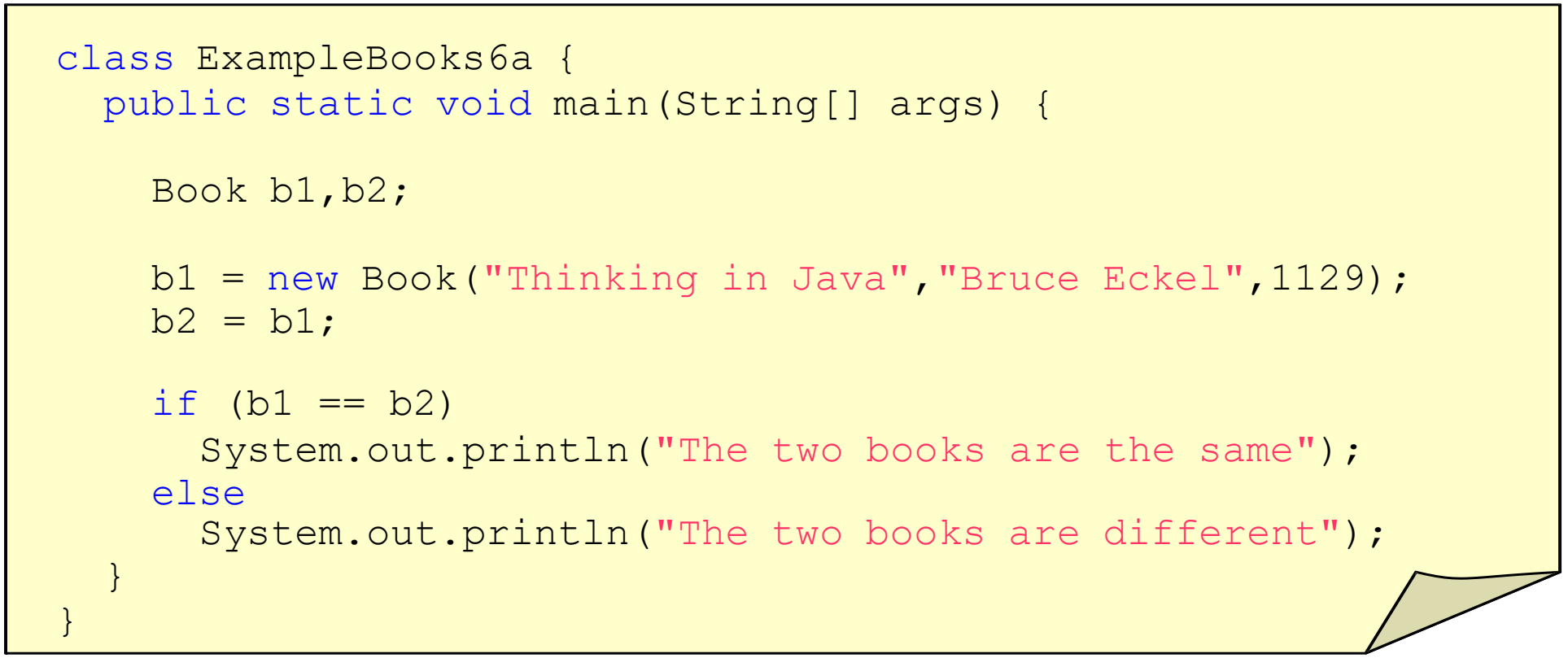
```
$ java ExampleBooks6
The two books are different
```

EQUALITY AND EQUIVALENCE




```
b1 = new Book("Thinking in Java","Bruce Eckel",1129);
b2 = new Book("Thinking in Java","Bruce Eckel",1129);

if (b1 == b2)
  System.out.println("The two books are the same");
else
  System.out.println("The two books are different");
```

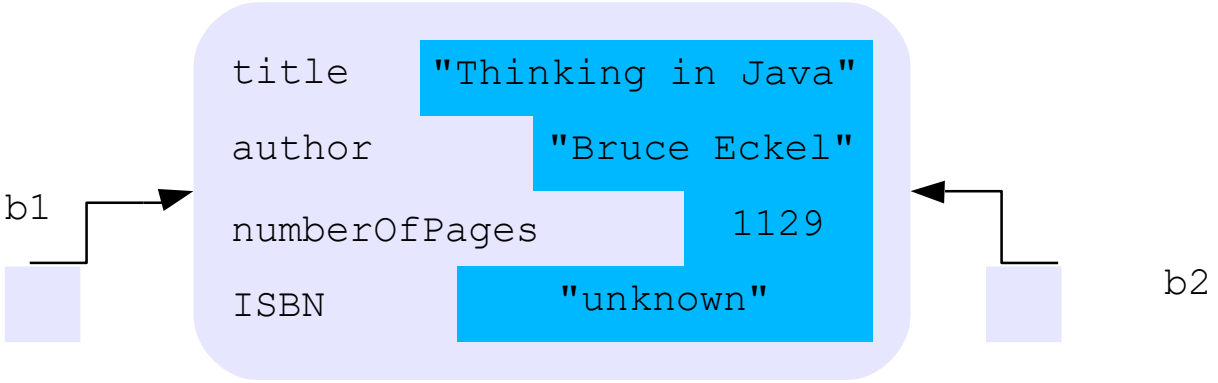
EQUALITY AND EQUIVALENCE



```
class ExampleBooks6a {
  public static void main(String[] args) {

    Book b1,b2;

    b1 = new Book("Thinking in Java","Bruce Eckel",1129);
    b2 = b1;

    if (b1 == b2)
      System.out.println("The two books are the same");
    else
      System.out.println("The two books are different");
  }
}
```

```
$ java ExampleBooks6a
The two books are the same
```

EQUALITY AND EQUIVALENCE

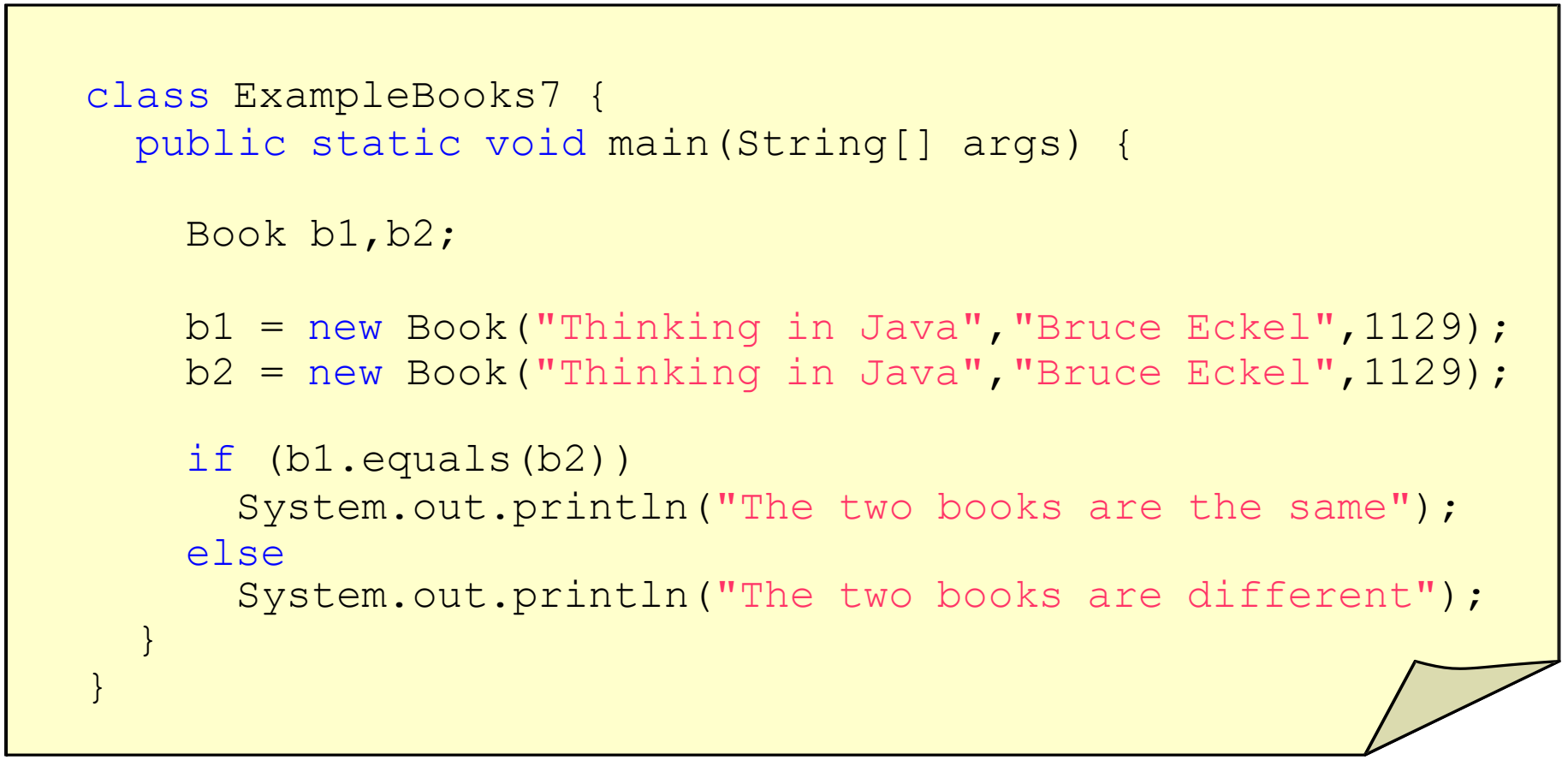



```
b1 = new Book("Thinking in Java","Bruce Eckel",1129);
b2 = b1;

if (b1 == b2)

  System.out.println("The two books are the same");
else
  System.out.println("The two books are different");
```

EQUALITY AND EQUIVALENCE

```
class Book {
  String title;
  String author;
  int numberOfPages;
  String ISBN;

  ...

  // compare two books
  public boolean equals(Book b) {
    return (title.equals(b.title) &&
            author.equals(b.author) &&
            numberOfPages == b.numberOfPages &&
            ISBN.equals(b.ISBN));
  }
}
```

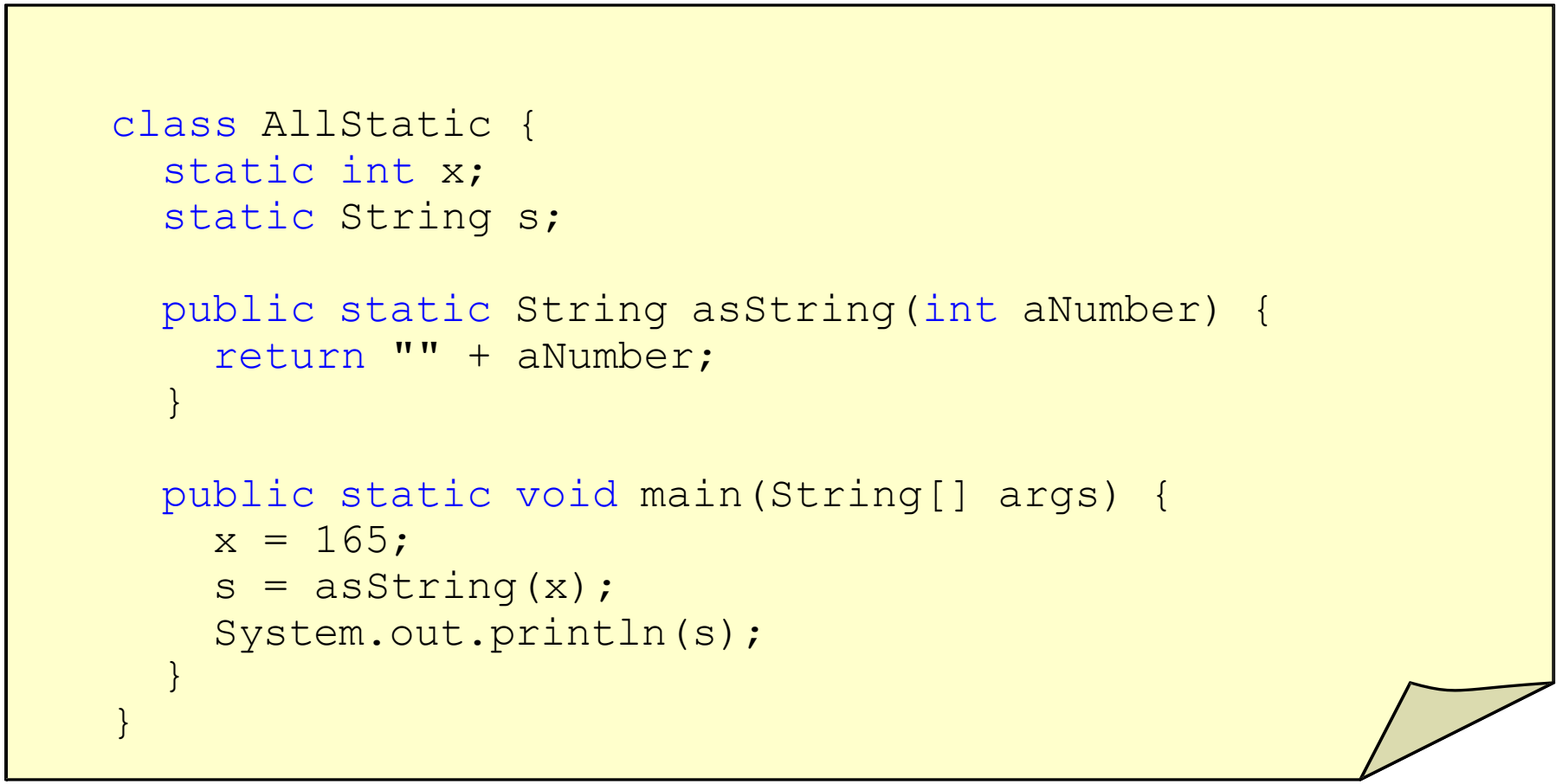
EQUALITY AND EQUIVALENCE



```
class ExampleBooks7 {
  public static void main(String[] args) {

    Book b1,b2;

    b1 = new Book("Thinking in Java","Bruce Eckel",1129);
    b2 = new Book("Thinking in Java","Bruce Eckel",1129);

    if (b1.equals(b2))
      System.out.println("The two books are the same");
    else
      System.out.println("The two books are different");
  }
}
```

```
$ java ExampleBooks7
The two books are the same
```

A STATIC APPLICATION



```
class AllStatic {
  static int x;
  static String s;

  public static String asString(int aNumber) {
    return "" + aNumber;
  }

  public static void main(String[] args) {
    x = 165;
    s = asString(x);
    System.out.println(s);
  }
}
```

```
$ java AllStatic
165
```

Deutsche Bank
Corporate Division

INSTANCE INITIATION

- All data members in an object are guaranteed to have an initial value.
- There exists a default value for all primitive types:

Types	Byte	Short	Int	Long	Float	Double	Char	Boolean	Reference
Initial Value	0	0	0	0	0.0F	0.0	'/0'	False	Null

Use this light-coloured cover template to save on toner when printing.

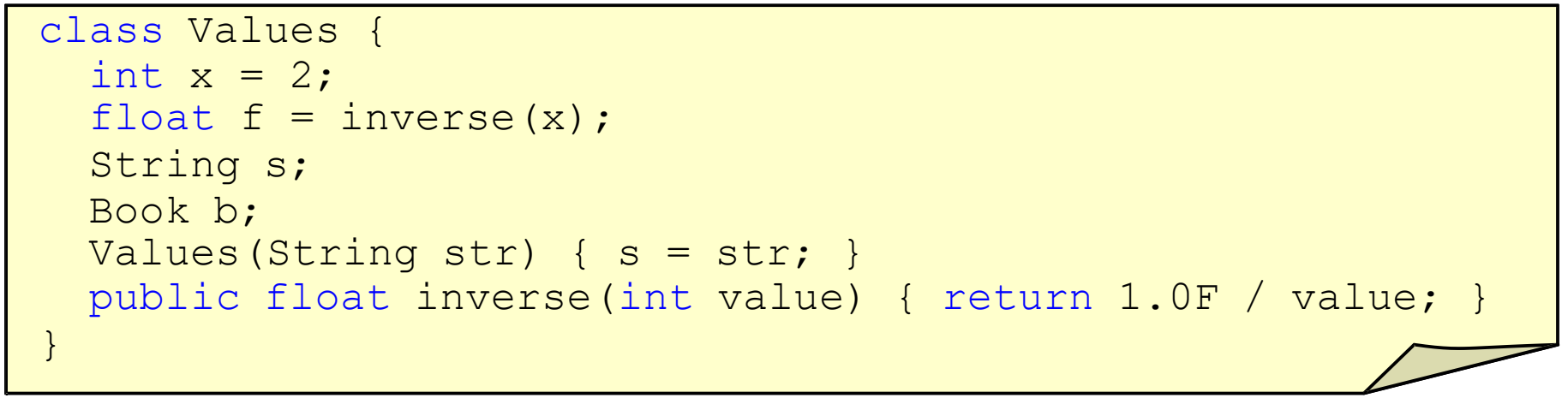
INSTANCE INITIATION

```
class Values {
  int x;
  float f;
  String s;
  Book b;
}
```

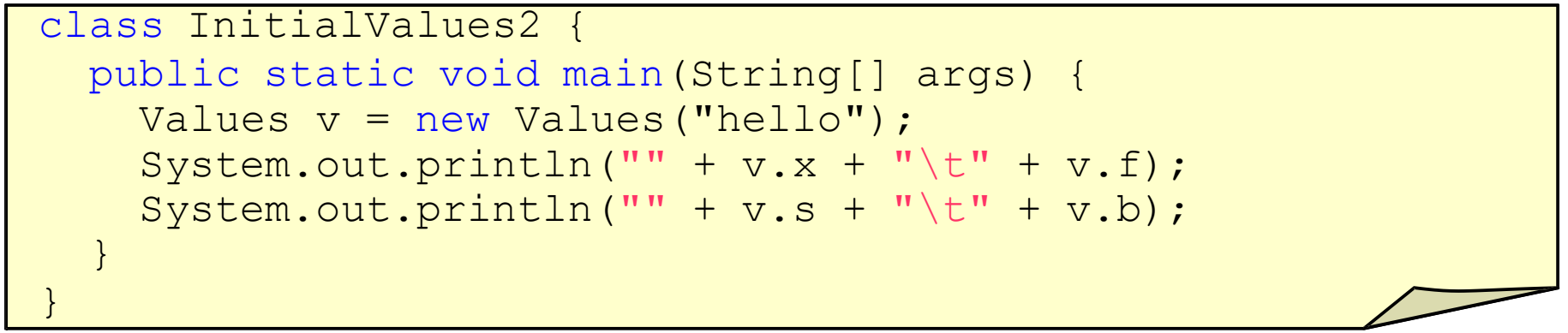
```
class InitialValues {
  public static void main(String[] args) {

    Values v = new Values();
    System.out.println(v.x);
    System.out.println(v.f);
    System.out.println(v.s);
    System.out.println(v.b);
  }
}
```

```
$ java InitialValues
0 0.0 null null
```

INSTANCE INITIATION



```
class Values {
  int x = 2;
  float f = inverse(x);
  String s;
  Book b;
  Values(String str) { s = str; }
  public float inverse(int value) { return 1.0F / value; }
}
```



```
class InitialValues2 {
  public static void main(String[] args) {
    Values v = new Values("hello");
    System.out.println("" + v.x + "\t" + v.f);
    System.out.println("" + v.s + "\t" + v.b);
  }
}
```

```
$ java InitialValues2
2 0.5
hello null
```

Deutsche Bank
Corporate Division

The keyword ‘this’

- The keyword this, when used inside a method, refers to the receiver object.
- It has two main uses:
  - to return a reference to the receiver object from a method
  - to call constructors from other constructors.

8 May 2024, Speaker name

THE KEYWORD 'this'

For example, the method setOwner in the previous Book class could have been defined as follows:

```
public Book setOwner(String name) {
   owner = name;
   return this;
}
```

```
Book b1 = new Book("Thinking in Java","Bruce Eckel",1129);
System.out.println(b1.setOwner("Carlos Kavka").getInitials());
System.out.println(b1.getOwner());
```

```
B.E.
Carlos Kavka
```

THE KEYWORD 'this'

The class Book has two constructors:

```
Book(String tit,String aut,int num) {
  title = tit; author = aut; numberOfPages = num;
  ISBN = "unknown";
}
Book(String tit,String aut,int num,String isbn) {
  title = tit; author = aut; numberOfPages = num;
  ISBN = isbn;
}
```

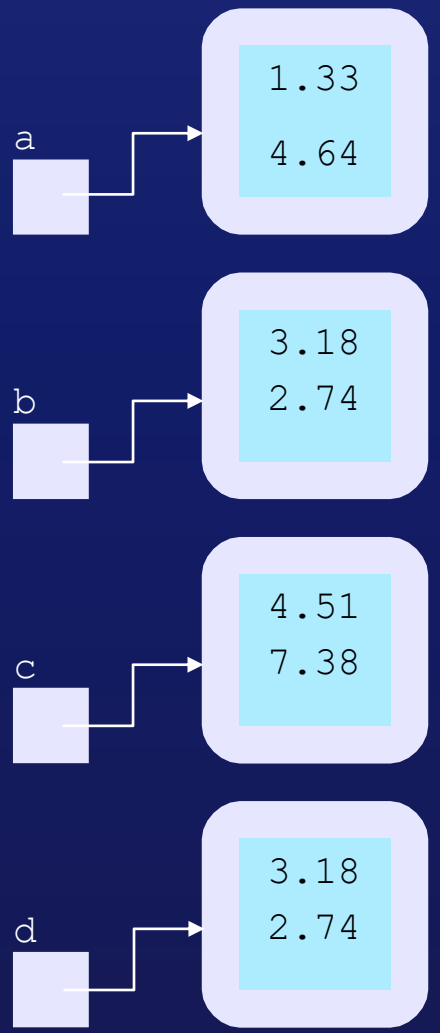
The second can be defined in terms of the first one:

```
Book(String tit,String aut,int num,String isbn) {
  this(tit,aut,num); ISBN = isbn;
}
```

AN EXAMPLE: COMPLEX CLASS

```
class TestComplex {

  public static void main(String[] args) {
    Complex a = new Complex(1.33,4.64);
    Complex b = new Complex(3.18,2.74);
    Complex c = a.add(b);

    System.out.println("c=a+b=" + c.getReal()
                + " " + c.getImaginary());

    Complex d = c.sub(a);
    System.out.println("d=c-a=" + d.getReal()
                + " " + d.getImaginary());
  }
}
```

```
$ java TestComplex
c=a+b= 4.51 7.38 d=c-a= 3.18 2.74
```



AN EXAMPLE: COMPLEX CLASS

```
class Complex {

  double real;      // real part
  double im;        // imaginary part

  Complex(double r,double i) {
    real = r;
    im = i;
  }

  public double getReal() {
    return real;
  }

  public double getImaginary() {
    return im;
  }
}
```

```
a = Complex(1.33,4.64)
```

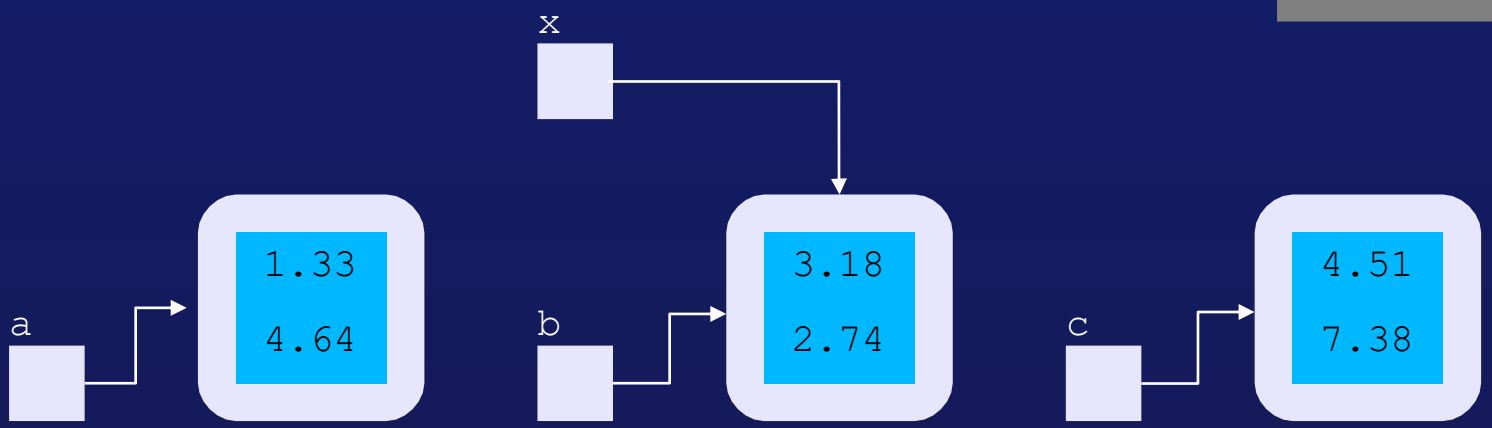
```
double realPart = a.getReal()
```

```
double imPart = a.getImmaginary()
```

AN EXAMPLE: COMPLEX CLASS

```
// add two complex numbers
public Complex add(Complex x) {
  return new Complex(real + x.real,im + x.im);
}
```

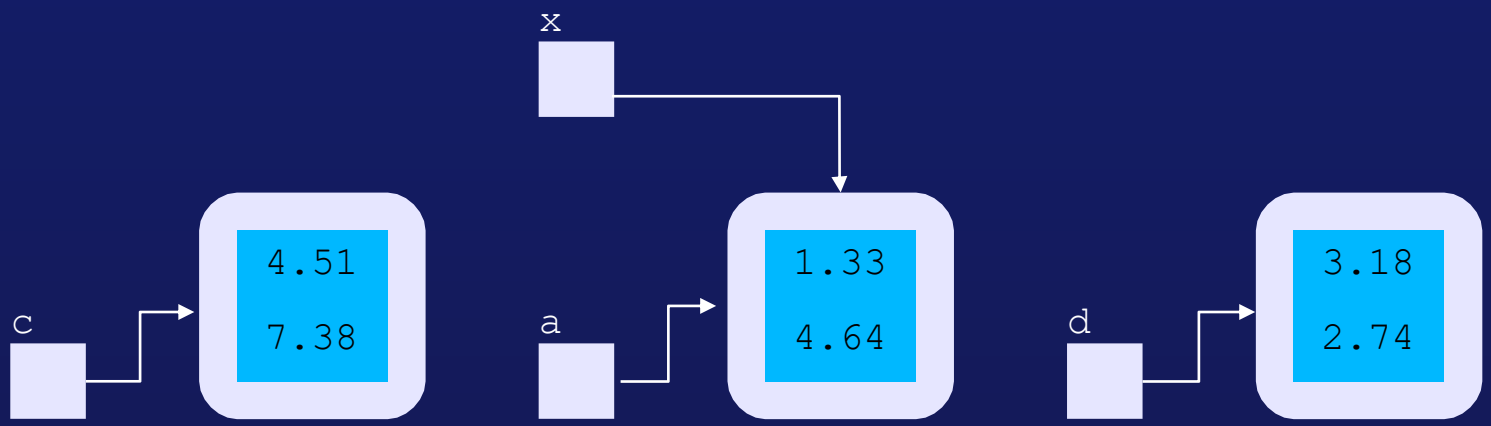
```
Complex c = a.add(b);
```



AN EXAMPLE: COMPLEX CLASS

```
// substract two complex numbers
public Complex sub(Complex c) {
   return new Complex(real - c.real,im - c.im);
}
```

```
Complex d = c.sub(a);
```

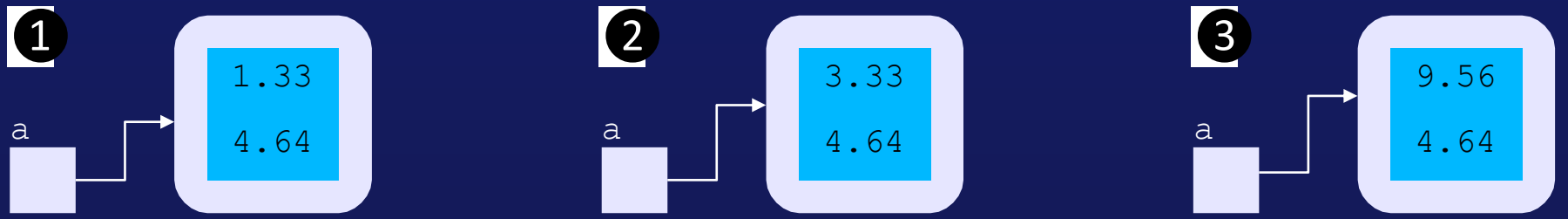


AN EXAMPLE: COMPLEX CLASS

The method **addReal** increments just the real part of the receptor of the message with the value passed as argument:

```
public Complex addReal(double x) {
   real += x;
   return this;
}
```

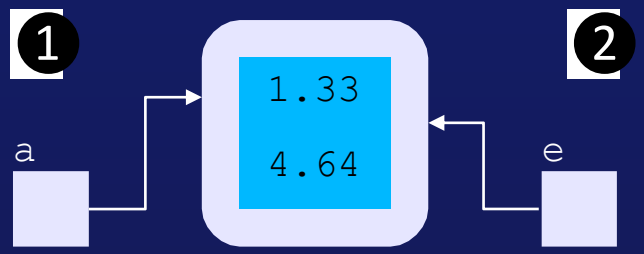
```
1 Complex a = new Complex(1.33,4.64);
2 a.addReal(2.0);
3 a.addReal(3.0).addReal(3.23);
```



AN EXAMPLE: COMPLEX CLASS

We must be careful if we want to create one complex number as a copy of the other:

```
1 Complex a = new Complex(1.33,4.64);
2 Complex e = a;
```



What will be the effect of

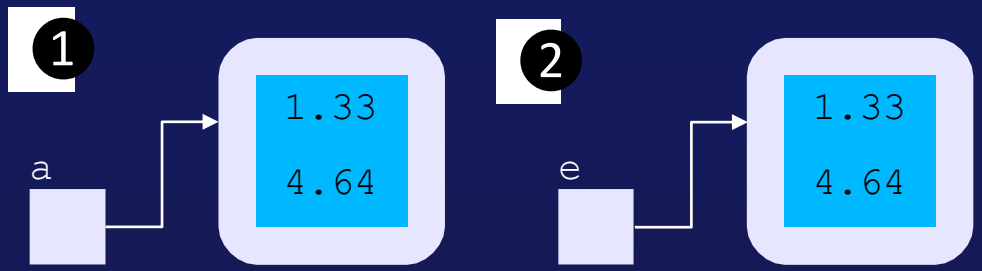
`e.addReal(5.6);` ?

AN EXAMPLE: COMPLEX CLASS

We can define a new constructor to avoid the problem:

```
Complex(Complex x) {
  this(x.real,x.im);
}
```

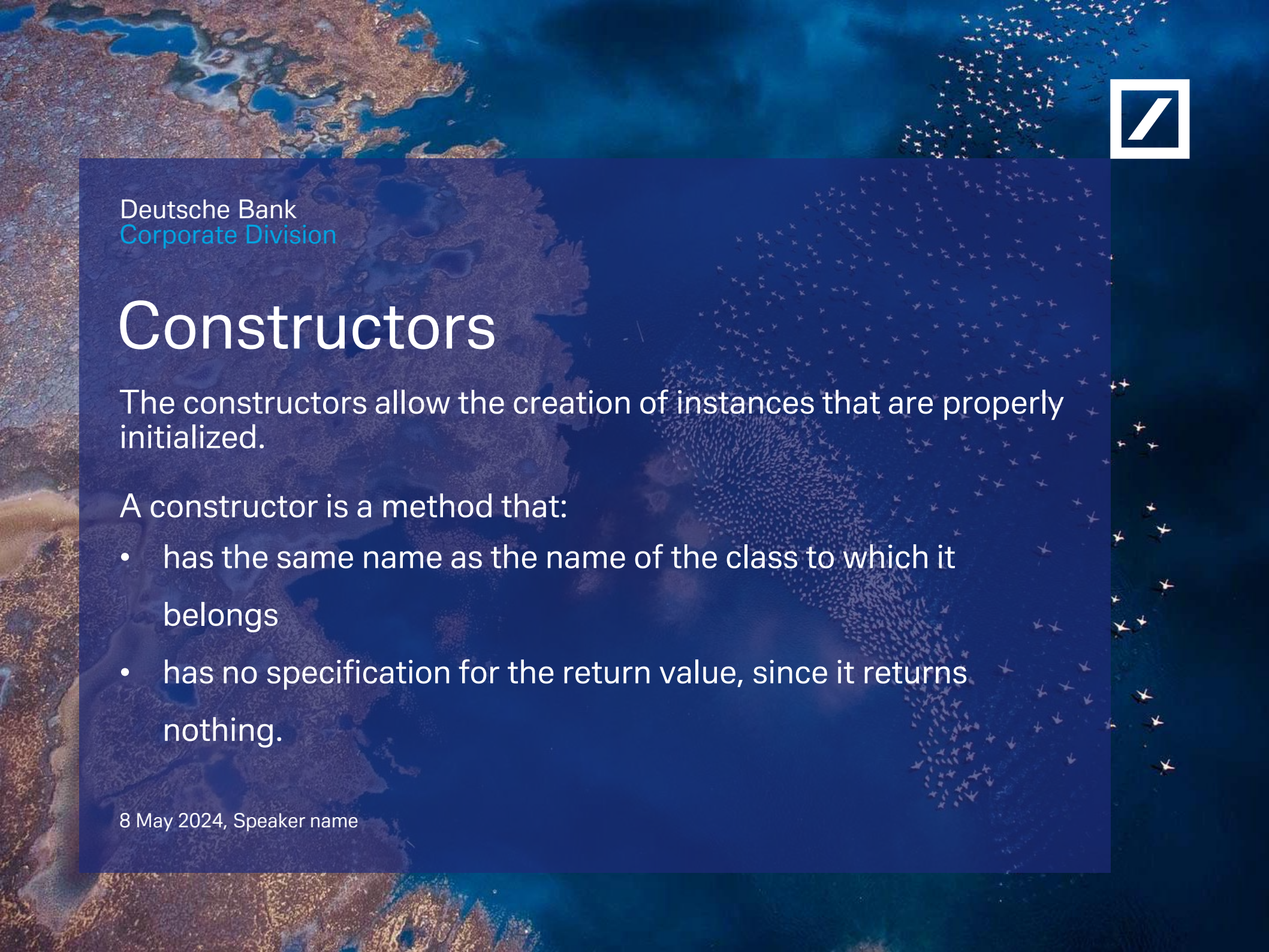
```
1 Complex a = new Complex(1.33,4.64);
2 Complex e = new Complex(a);
```



Deutsche Bank
Corporate Division

CONSTRUCTORS

- Definition
- Default Constructors
- Multiple Constructors



Deutsche Bank
Corporate Division

Constructors

The constructors allow the creation of instances that are properly initialized.

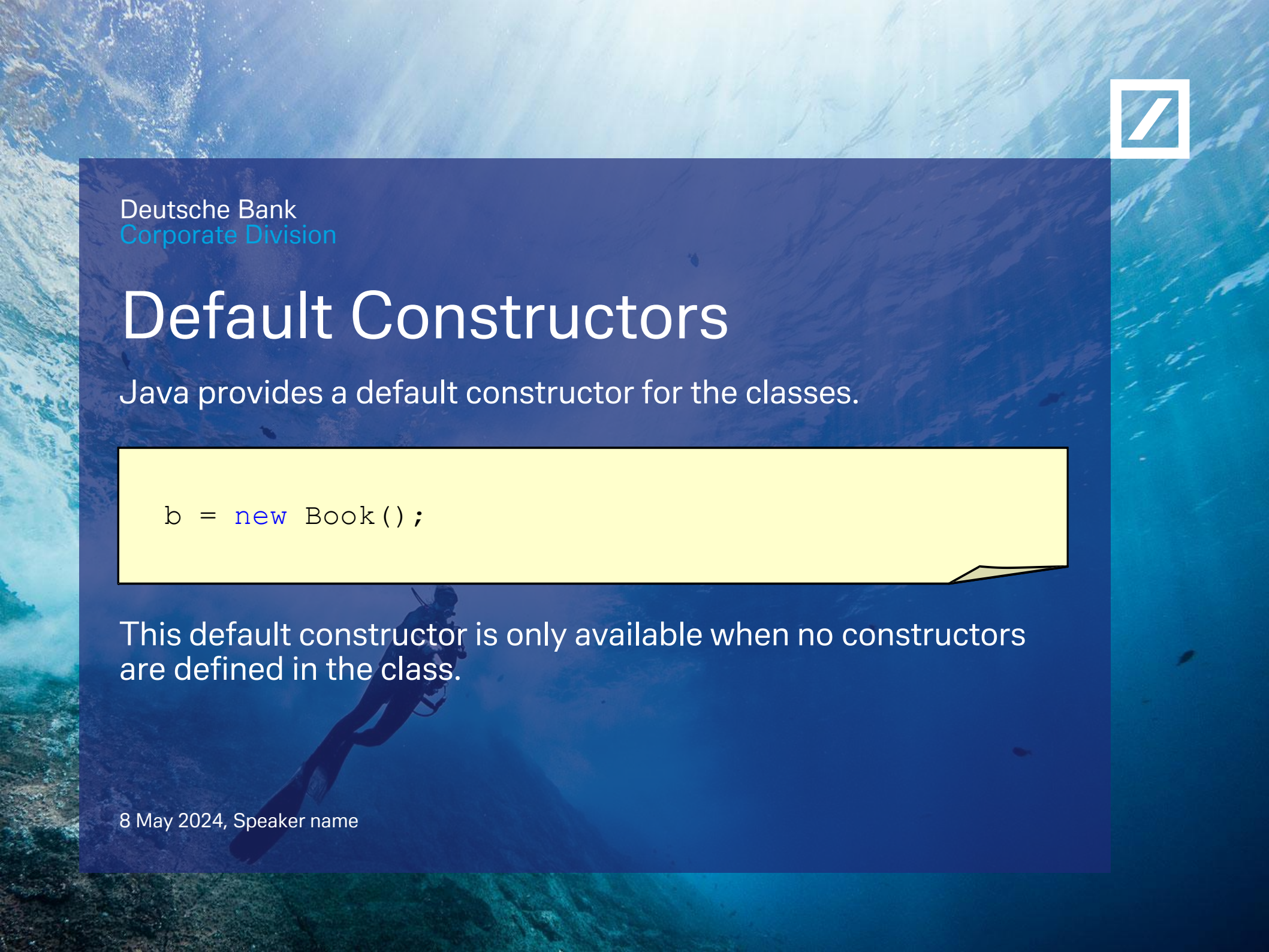
A constructor is a method that:

- has the same name as the name of the class to which it belongs
- has no specification for the return value, since it returns nothing.

8 May 2024, Speaker name

```
class Book {
  String title;
  String author;
  int numberOfPages;
  Book(String tit,String aut,int num) {
    title = tit;
    author = aut;
    numberOfPages = num;
  }
}
```

```
class ExampleBooks2 {
  public static void main(String[] args) {
  Book b = new Book("Thinking in Java","Bruce Eckel",1129);
  System.out.println(b.title + " : " + b.author +
                     " : " + b.numberOfPages);
  }
}
```



Deutsche Bank
Corporate Division

Default Constructors

Java provides a default constructor for the classes.

```
b = new Book();
```

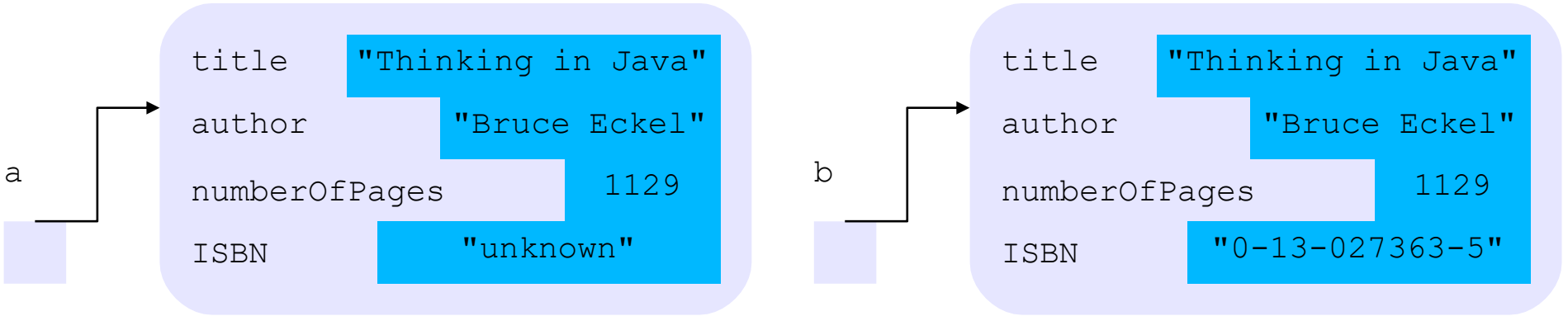
This default constructor is only available when no constructors are defined in the class.

8 May 2024, Speaker name

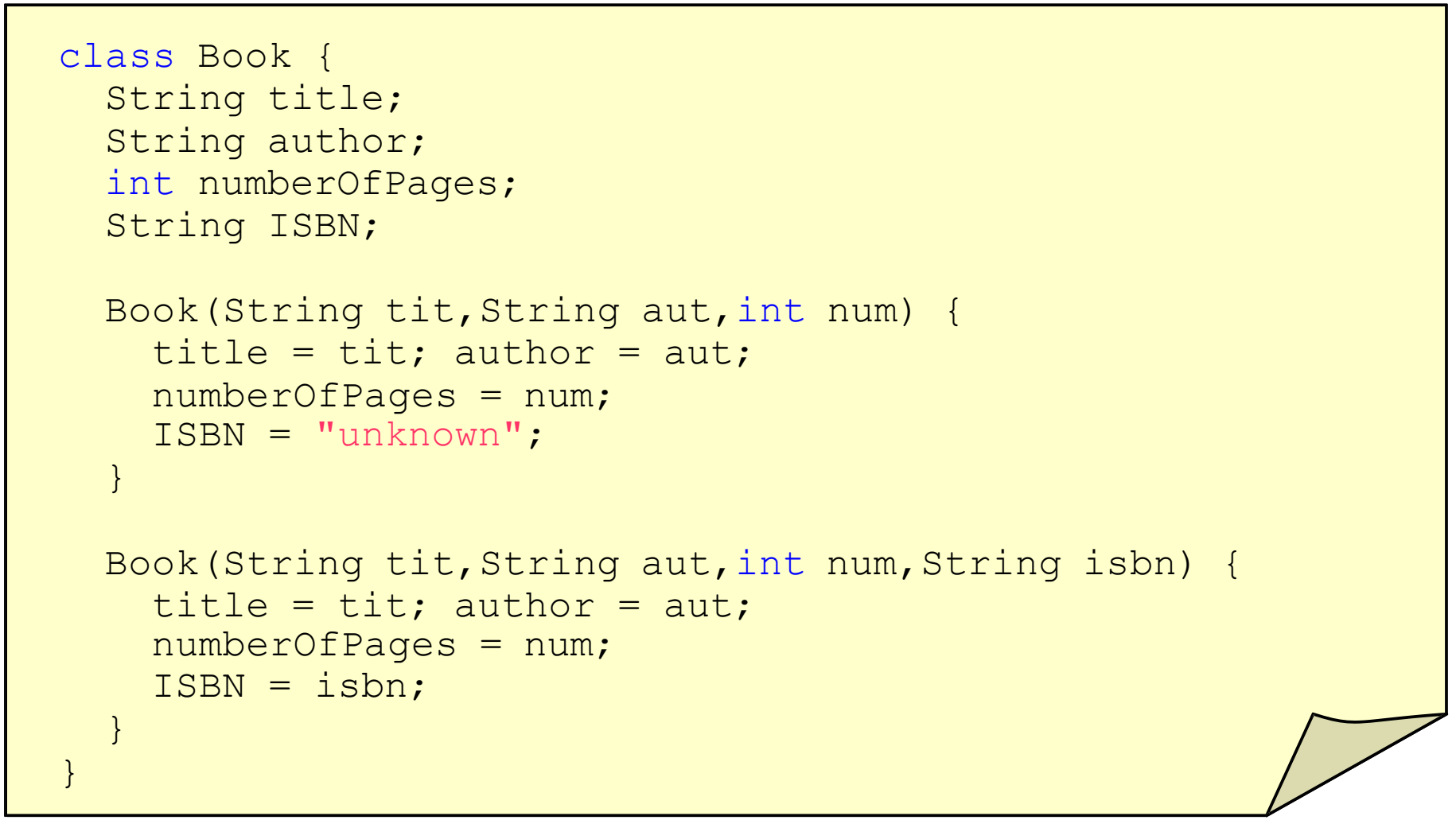
MULTIPLE CONSTRUCTORS

It is possible to define more than one constructor for a single class, only if they have different number of arguments or different types for the arguments.

```
a = new Book("Thinking in Java","Bruce Eckel",1129);
b = new Book("Thinking in Java","Bruce Eckel",1129,"0-13-027363");
```



MULTIPLE CONSTRUCTORS

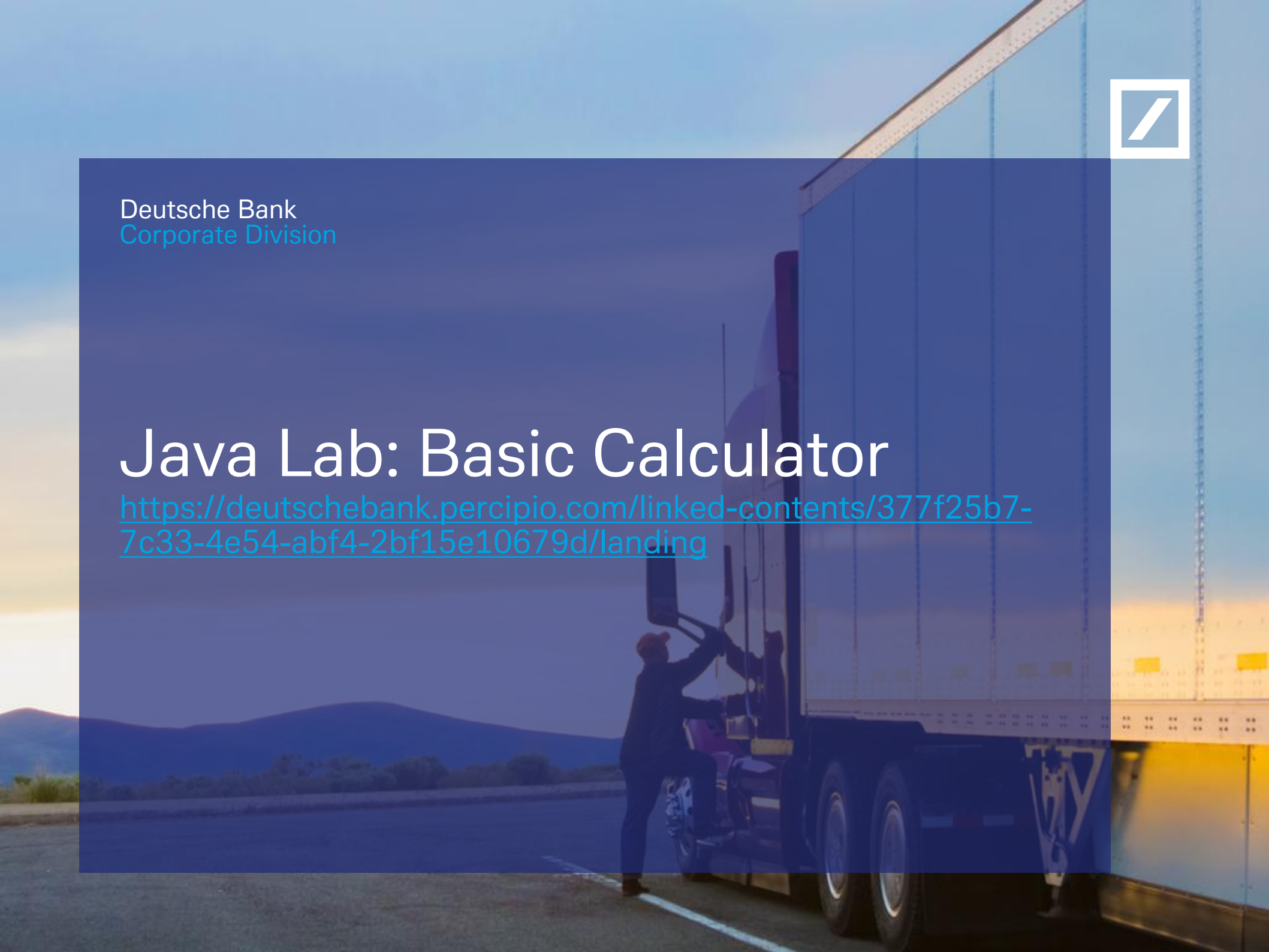


```
class Book {
  String title;
  String author;
  int numberOfPages;
  String ISBN;

  Book(String tit,String aut,int num) {
    title = tit; author = aut;
    numberOfPages = num;
    ISBN = "unknown";
  }

  Book(String tit,String aut,int num,String isbn) {
    title = tit; author = aut;
    numberOfPages = num;
    ISBN = isbn;
  }
}
```

MULTIPLE CONSTRUCTORS

```
class ExampleBooks3 {
  public static void main(String[] args) {
    Book b1,b2;

    b1 = new Book("Thinking in Java","Bruce Eckel",1129);
    System.out.println(b1.title + " : " + b1.author +
                       " : " + b1.numberOfPages + " : " + b1.ISBN);
    b2 = new Book("Thinking in Java","Bruce Eckel",1129,
                        "0-13-027363-5");
    System.out.println(b2.title + " : " + b2.author +
                       " : " + b2.numberOfPages + " : " + b2.ISBN);
  }
}
```

```
$ java ExampleBooks3
Thinking in Java : Bruce Eckel : 1129 : unknown
Thinking in Java : Bruce Eckel : 1129 : 0-13-027362-5
```



Deutsche Bank
Corporate Division

Java Lab: Basic Calculator

https://deutschebank.percipio.com/linked-contents/377f25b7-7c33-4e54-abf4-2bf15e10679d/landing

Deutsche Bank
Corporate Division

Java Lab: Bug Detective

https://deutschebank.percipio.com/linked-contents/95804221-e862-4870-ac4a-12ffeb9cf501/landing

Deutsche Bank
Corporate Division

Java Lab: DNA Sequencing

https://deutschebank.percipio.com/linked-contents/30b25c91-4b21-44c6-a624-6b3e32185322/landing

Deutsche Bank
Corporate Division

Java Lab: Fizz Buzz

https://deutschebank.percipio.com/linked-contents/f5f06e35-9ee0-4737-9be0-10b8665b85b8/landing


Deutsche Bank
Corporate Division
Java Lab: Language Families
https://deutschebank.percipio.com/linked-contents/cf645a03-7976-4617-abd3-2754ef778a60/landing

Deutsche Bank
Corporate Division

Java Lab: Loan Payment Calculator

(Optional) - https://deutschebank.percipio.com/linked-contents/24ecae32-3851-40d7-8a76-0dad365c2070/landing

Deutsche Bank
Corporate Division

Java Variables Lab: Mad Libs

(Optional) - https://deutschebank.percipio.com/linked-contents/829a740c-9614-48a8-b6f2-7609f9d312dc/landing