Learn more about our research, discover data science, and find other great resources at:

http://www.dataminingapps.com

Chapter 4
Moving Toward Object-Oriented Programming

Overview

- Classes and Objects in Java
- Storing Data: Variables
- Defining Behavior: Methods
- Java SE Built-in Classes

Classes and Objects in Java

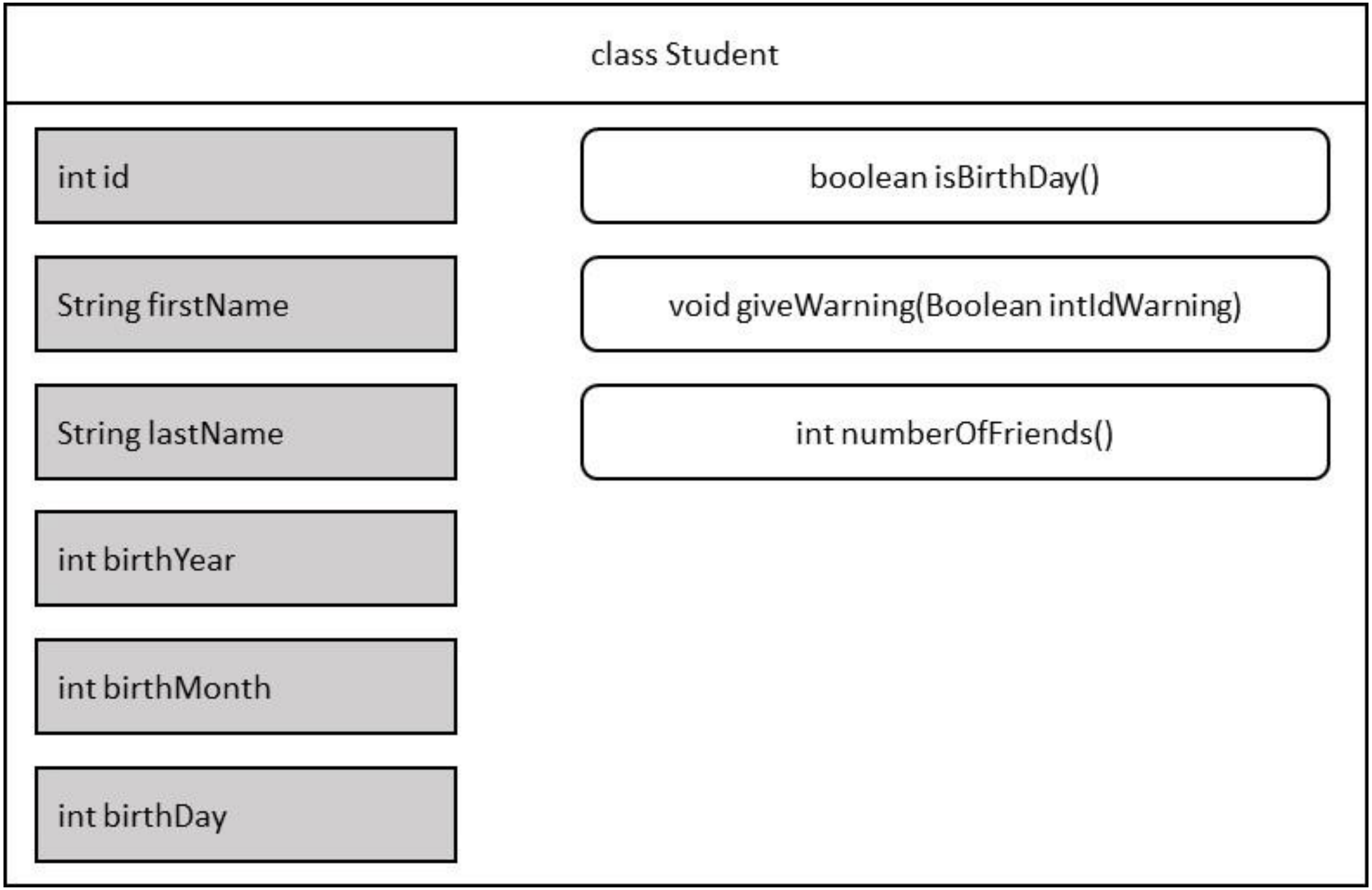
- Class is a blueprint or template of object characteristics
- Object is an instance of a class
- Objects encapsulate both data and behavior
  - Data: variables
  - Behavior: methods
- Data can only be accessed by means of methods (information hiding)

Classes and Objects in Java

- Java is a pure OO language
  - Autoboxing: primitive data types (boolean, byte, int, double, …) have corresponding wrapper classes (Boolean, Byte, Integer, Double, …)
- Java class is defined as follows:

```
class CLASSNAME {
// VARIABLE DEFINITIONS
// METHOD DEFINITIONS
}
```

Classes and Objects in Java



class Student

int id

String firstName

String lastName

int birthYear

int birthMonth

int birthDay

boolean isBirthDay()

void giveWarning(Boolean intIdWarning)

int numberOfFriends()

Classes and Objects in Java

```java
class Student {
int id;
String firstName;
String lastName;
int birthYear, birthMonth, birthDay;

boolean isBirthday() {
// Return true if it's the student's birthday today.
return false;
}
void giveWarning(boolean isFinalWarning) {
// You should study harder!
}
int numberOfFriends() {
// Return the number of friends the student has.
return 0;
}
}
```

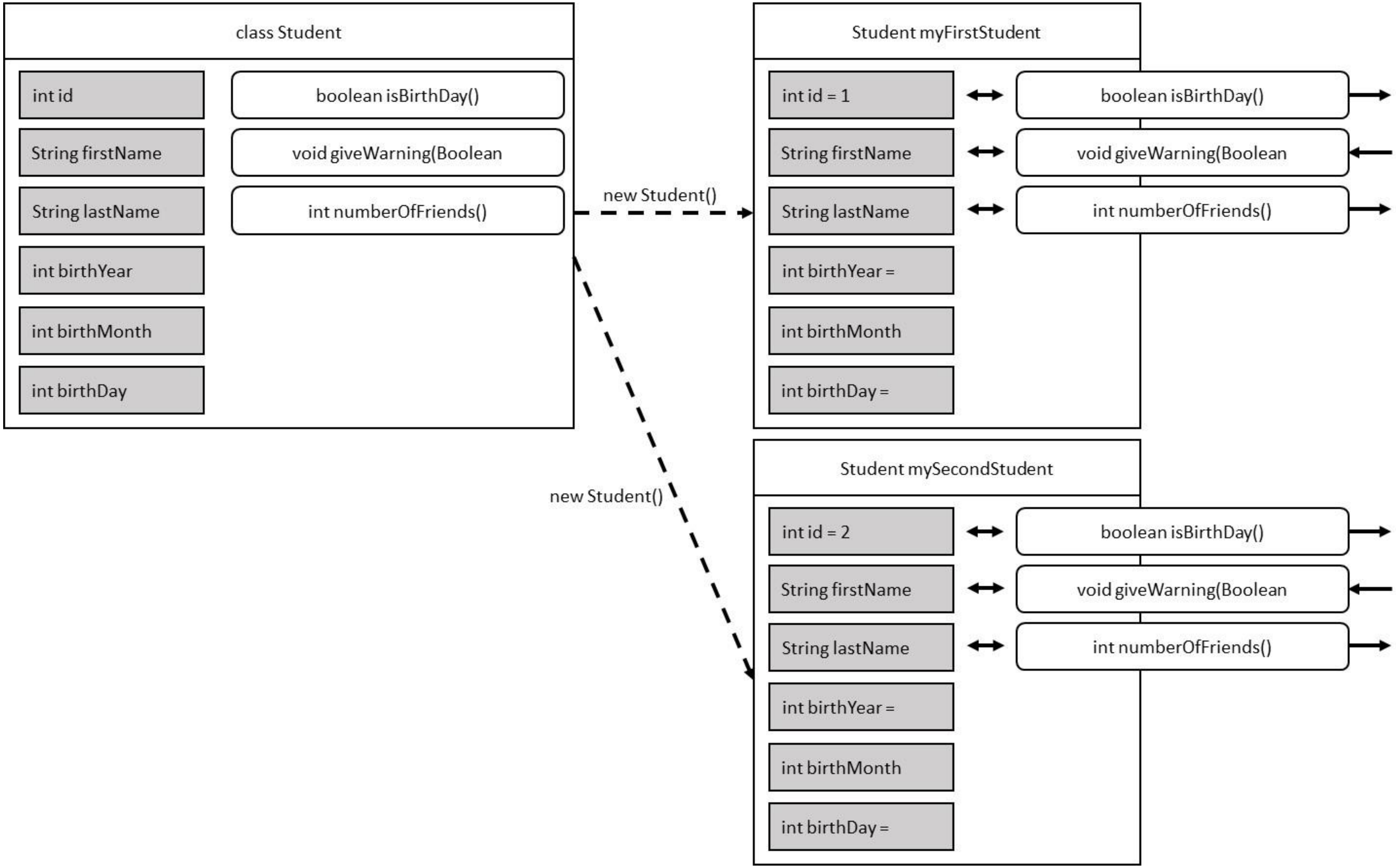
Variables

Methods

Classes and Objects in Java

- Objects are an instance of a class
- Objects are created using the `new` keyword
  - `Student myFirstStudent= new Student();`

Classes and Objects in Java



Classes and Objects in Java

```
class Student {
…
public static void main(String[] args) {
   Student firstStudent = new Student();
   Student secondStudent = new Student();
   firstStudent.id = 1;
   firstStudent.firstName = "Marc";
   secondStudent.id = 2;
   secondStudent.firstName = "Sophie";
   System.out.println("The first name of the secondStudent is:" +
   secondStudent.firstName);
   }
   }
```

Storing Data: Variables

- Instance variables
- Class variables
- Final variables
- Variable scope

Instance Variables

- Instance variables belong to objects (aka member variables, fields)
- Instance variables are assigned default values upon definition

Instance Variables

```
public class Book {

String title = "Unknown Title";
String[] authors = new String[]{"Anonymous"};
int yearReleased = 2014, copiesSold = 0;

public static void main(String[] args) {
Book book1, book2;
book1 = new Book(); // Set first book
book1.title = "Beginning Java Programming";
book1.authors = new String[]{"Bart Baesens","Aimee Backiel","Seppe vanden Broucke"};

book2 = new Book(); // Set second book
book2.title = "Catcher in the Rye";
book2.authors = new String[]{"J. D. Salinger"};
}
}
```

Class Variables

- A class variable is shared between all objects belonging to the class (aka static variable)
- Can be accessed by means of the class or object

Class Variables

```
public class Book {

static int maxAmountOfPages=500;

String title = "Unknown Title";
String[] authors = new String[]{"Anonymous"};
int yearReleased = 2014, copiesSold = 0, nrOfPages=1400;

public static void main(String[] args) {
Book superLargeBook = new Book();
superLargeBook.title = "Super Large Boring Book";
System.out.println("I have a book here with the title: "+superLargeBook.title);
System.out.println("Written by: "+superLargeBook.authors);
System.out.println("Released in: "+superLargeBook.yearReleased);
System.out.println("With number of pages: "+superLargeBook.nrOfPages);
System.out.println("However, we only support books with max. pages: “
+superLargeBook.maxAmountOfPages);
System.out.println("However, we only support books with max. pages: "
+Book.maxAmountOfPages);
}}
```

Final Variables

- Final variables are initialized only once!

```
public class Book {
final String title = "Unknown Title";
String[] authors = new String[]{"Anonymous"};
int yearReleased = 2014, copiesSold = 0, nrOfPages;

public static void main(String[] args) {
Book superLargeBook = new Book();
superLargeBook.title = "Super Large Boring Book";
superLargeBook.nrOfPages = 1400;
}
}
```

Final Variables

```
class Book {
final static int MAX_AMOUNT_OF_PAGES = 500;
final static int MIN_AMOUNT_OF_PAGES = 50;
String title;
String[] authors;
int yearReleased, nrOfPages;
int copiesSold = 0;

public static void main(String[] args) {
Book superLargeBook = new Book();
superLargeBook.title = "Super Large Boring Book";
superLargeBook.nrOfPages = 1400;
System.out.println("Check if your book has a correct amount of pages...");
System.out.println("- Minimum amount: "+Book.MIN_AMOUNT_OF_PAGES);
System.out.println("- Maximum amount: "+Book.MAX_AMOUNT_OF_PAGES);
System.out.println("- Your book: "+superLargeBook.nrOfPages);
}
}
```

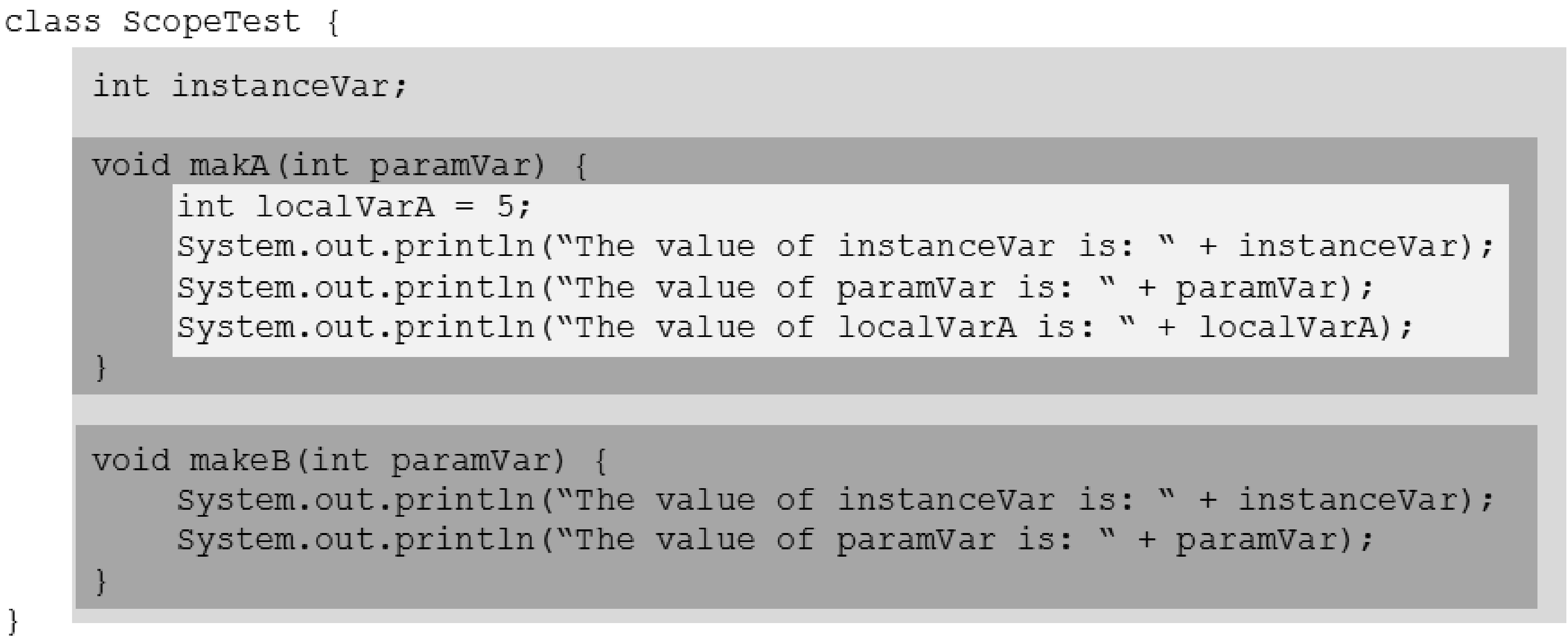
Final variables

```
class Book {
final static int MAX_AMOUNT_OF_PAGES = 500;
final static int MIN_AMOUNT_OF_PAGES = 50;
String title;
String[] authors;
int yearReleased, nrOfPages;
int copiesSold = 0;

public static void main(String[] args) {
final Book superLargeBook = new Book();
superLargeBook.title = "Super Large Boring Book";
superLargeBook.nrOfPages = 1400;
// Change the amount of copies sold
superLargeBook.copiesSold += 1000;
// Assign a new book
superLargeBook = new Book(); // EEK!
}
}
```

Variable Scope

- A variable’s scope is the context in which the variable is known
- Levels:
  - Local variable: declared inside a method or block
  - Parameter variable: declared as a method argument or loop variable
  - Instance variable: declared in the class definition

Variable Scope



```
class ScopeTest {
    int instanceVar;

    void makA(int paramVar) {
        int localVarA = 5;
        System.out.println(“The value of instanceVar is: “ + instanceVar);
        System.out.println(“The value of paramVar is: “ + paramVar);
        System.out.println(“The value of localVarA is: “ + localVarA);
    }

    void makeB(int paramVar) {
        System.out.println(“The value of instanceVar is: “ + instanceVar);
        System.out.println(“The value of paramVar is: “ + paramVar);
    }
}
```

Variable Scope

```
class ScopeTest {

void makeA() {
int a = 5;
}

void readA() {
System.out.println("The value of a is: "+a);

}
}
```

Variable Scope

```
class ScopeTest {

int a = 5;

void printA() {
int a = 10;
System.out.println("The value of a is now: "+a);

}
}
```

Defining Behavior: Methods

- Instance Methods
- Class Methods
- Constructors
- `main` Method
- Method Argument Passing

Instance Methods

- Instance method is a method that is accessible only through initialized objects (aka member method)

Instance Methods

```
public class Dog {
boolean isSitting;

String getBarkSound() {
return "Woof!";}

boolean isSitting() {
return isSitting;}


void sit() {
isSitting = true;}

void stand() {
isSitting = false;}

public static void main(String[] args) {
Dog myDog = new Dog();
// Call the instance method on the object myDog:
System.out.println(myDog.getBarkSound());


}}
```

Instance Methods

```
void giveCookie(Cookie cookie) { /*...*/ }
void chaseDog(Dog dog) { /*...*/ }
void lickPerson(Person person, int nrLicks) { /*...*/ }
void giveNickNames(String[] nickNames) { /*...*/ }

String[] newNames = new String[] {"Puppers", "Droopy"};
myDog.giveNickNames(newNames);
```

Instance Methods

- Varargs (…) represents a variable number of arguments

```
void giveNickNames(String... nickNames) { /*...*/ }

void giveNickNames(String... nickNames) {
System.out.println("You have given me "+nickNames.length+" names");
}
myDog.giveNickNames("Puppers", "Droopy");
// Or any other amount of strings:
myDog.giveNickNames("Puppers", "Droopy", "Tissues", "Clifford");
myDog.giveNickNames();
```

Class Methods

- Class method is shared between all objects belonging to the class (aka static method)
- Class method doesn’t need object to be used

Class methods

```
public class Cat {

static String preferredFood() {
return "Fish";
}

public static void main(String[] args) {
Cat myCat = new Cat();
System.out.println("A cat's preferred food is: "+myCat.preferredFood());
System.out.println("A cat's preferred food is: "+Cat.preferredFood());
}
}
```

Class methods

```
class Cat {
String name;

static void changeName() {
name = "ANONYMOUS CAT";
}
}
```

Class Methods

```
class Cat {
static String preferredFood = "fish";

static String getPreferredFood() {
return preferredFood;
}

static void setPreferredFood(String newFood) {
preferredFood = newFood;
}
}
```

Constructors

- Special class method used to initialize objects of a class
- Defined similarly as an instance method, with the same name of the class and no return type (not even void)
- Java automatically assumes a blank constructor when you do not define one
- Constructor is invoked when you create a new object using the keyword new

Constructors

```
class Book {
final static int DEFAULT_YEAR = 2014;
final String title;
final int releaseYear;
int copiesSold;

Book(String t) {
title = t;
releaseYear = DEFAULT_YEAR;
// copiesSold will default to 0
}
Book(String t, int r) {
title = t;
releaseYear = r;
// copiesSold will default to 0
}
Book(String t, int r, int s) {
title = t;
releaseYear = r;
copiesSold = s;
}}
```

Constructors

```
class Book {
final static int DEFAULT_YEAR = 2014;
final String title;
final int releaseYear;
int copiesSold;

Book(String t) {        ←――――――― Book mybook=new Book(“Java Programming”);
title = t;
releaseYear = DEFAULT_YEAR;
// copiesSold will default to 0
}
Book(String t, int r) { ←――――――― Book mybook=new Book(“Java Programming”, 2015);
title = t;
releaseYear = r;
// copiesSold will default to 0
}
Book(String t, int r, int s) { ←―― Book mybook=new Book(“Java Programming”, 2015, 5000);
title = t;
releaseYear = r;
copiesSold = s;
}}
```

Constructors

```
class Book {
final static int DEFAULT_YEAR = 2014;
final String title;
final int releaseYear;
int copiesSold;

Book(String t) {
// Call other constructor:
this(t, DEFAULT_YEAR, 0);
}

Book(String t, int r) {
// Call other constructor:
this(t, r, 0);
}

Book(String t, int r, int s) {
title = t;
releaseYear = r;
copiesSold = s;
}}
```

`main` Method

- Method used as entry point to execute your program
- `public static void main(String[] args) {}`
- Ideally, a main method is not put into a class definition relating to a real-word concept but in a controller class

`main` Method

```java
class Book {
final String title;
final int releaseYear;
int copiesSold;

Book(String t, int r) {
title = t;
releaseYear = r;
}

void sell(int nrCopies) {
copiesSold += nrCopies;
}

int nrCopiesSold() {
return copiesSold;
}

}
```

```java
// File Program.java:
class Program {

public static void main(String[] args) {

Book firstBook = new Book("First Book", 2004);
Book secondBook = new Book("Another Book", 2014);

firstBook.sell(200);
System.out.println("Number of copies sold of
first book is now: " +firstBook);
System.out.println("Title of the second book is:
"+secondBook.title);
}
}
```

Method Argument Passing

- Primitive data types (e.g. boolean, int, double, …) are passed by value
- Non-primitive data types are passed by value of memory address of object

Method Argument Passing

```
class Test {
int a = 4;

static void increaseInt(int anInt) {
anInt++;
}

public static void main(String[] args) {
Test t = new Test();
System.out.println("Instance var a is: "+t.a);
Test.increaseInt(t.a);
System.out.println("Instance var a is now: "+t.a);
}
}
```

Method Argument Passing

```
class Test {
int a = 4;

static void increaseInt(int anInt) {
anInt++;
}

public static void main(String[] args) {
Test t = new Test();
System.out.println("Instance var a is: "+t.a);
Test.increaseInt(t.a);
System.out.println("Instance var a is now: "+t.a);
}
}
```

Output:

Instance var a is: 4

Instance var a is now: 4

Method Argument Passing

```
class Test {
int[] array = new int[]{1,2,3};

static void increaseFirstInt(int[] anIntArray) {
anIntArray[0]++;
}

public static void main(String[] args) {
Test t = new Test();
System.out.println("First element in array is: "+t.array[0]);
Test.increaseFirstInt(t.array);
System.out.println("First element in array is now:
"+t.array[0]);
}}
```

Method Argument Passing

```
class Test {
int[] array = new int[]{1,2,3};

static void increaseFirstInt(int[] anIntArray) {
anIntArray[0]++;
}

public static void main(String[] args) {
Test t = new Test();
System.out.println("First element in array is: "+t.array[0]);
Test.increaseFirstInt(t.array);
System.out.println("First element in array is now: "+t.array[0]);
}}
```

Output:

```
First element in array is: 1
First element in array is now: 2
```

Method Argument Passing

```
class Test {
int[] array = new int[]{1,2,3};

static void increaseFirstInt(int[] anIntArray) {
anIntArray[0]++;
}
static void changeIntArray(int[] anIntArray) {
anIntArray = new int[] {100,200,300};
}

public static void main(String[] args) {
Test t = new Test();
System.out.println("First element in array is: "+t.array[0]);
Test.increaseFirstInt(t.array);
System.out.println("First element in array is now: "+t.array[0]);
Test.changeIntArray(t.array);
System.out.println("First element in array is now: "+t.array[0]);
}}
```

Method Argument Passing

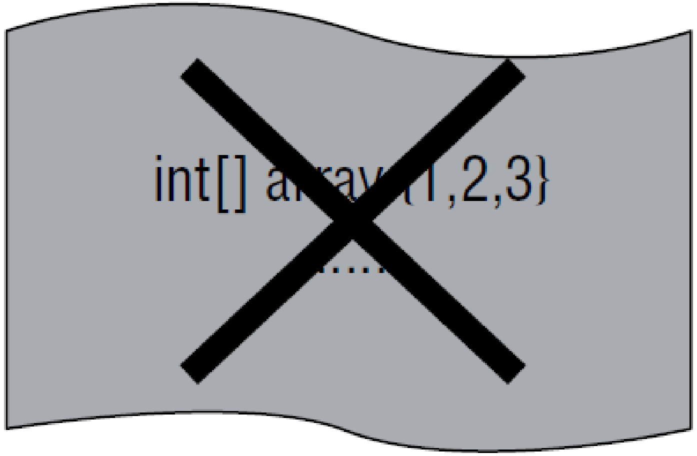
```
class Test {
int[] array = new int[]{1,2,3};

static void increaseFirstInt(int[] anIntArray) {
anIntArray[0]++;
}
static void changeIntArray(int[] anIntArray) {
anIntArray = new int[] {100,200,300};
}

public static void main(String[] args) {
Test t = new Test();
System.out.println("First element in array is: "+t.array[0]);
Test.increaseFirstInt(t.array);
System.out.println("First element in array is now: "+t.array[0]);
Test.changeIntArray(t.array);
System.out.println("First element in array is now: "+t.array[0]);
}}
```

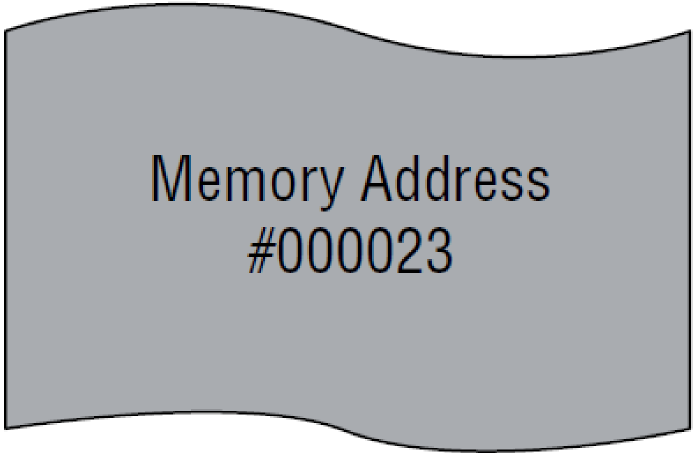
Output:

```
First element in array is: 1
First element in array is now: 2
First element in array is now: 2
```

Method Argument Passing

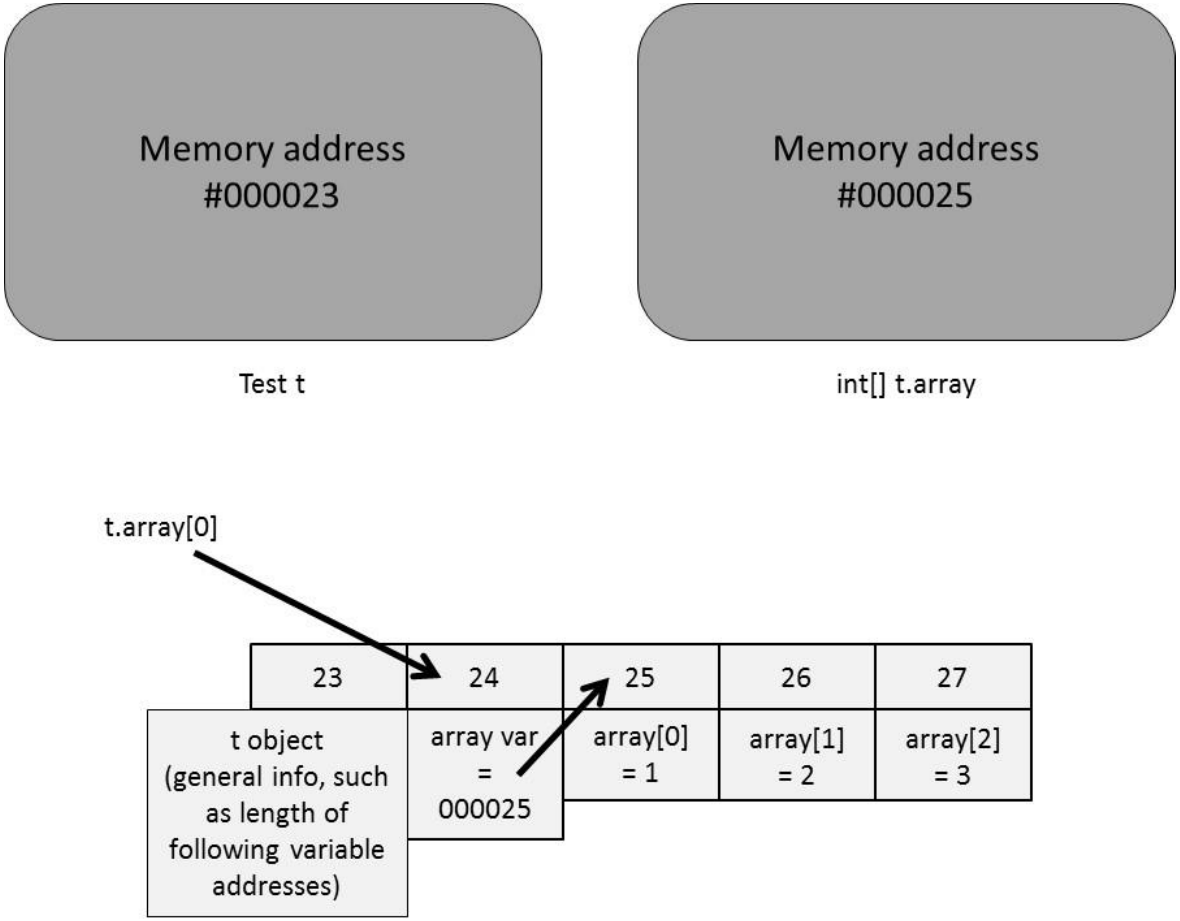
```
Test t = new Test();
```





Method Argument Passing

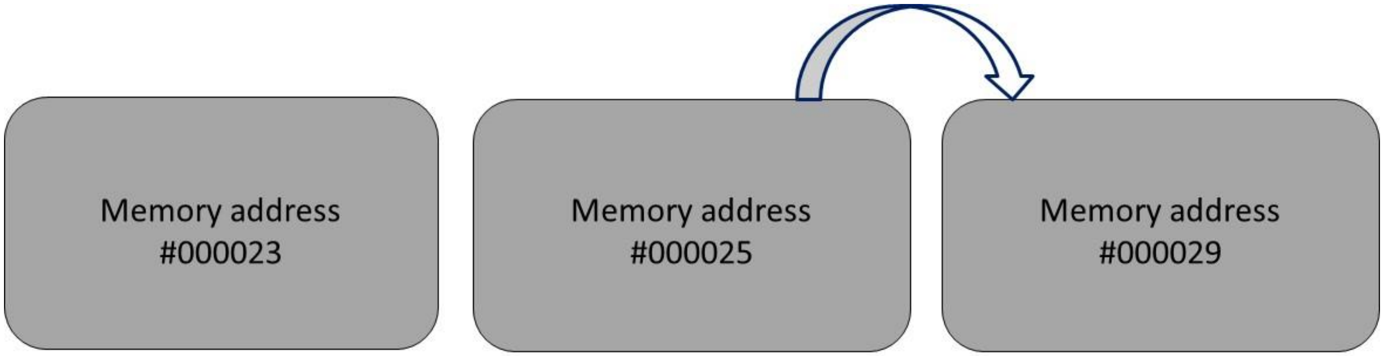
```
Test.increaseFirstInt(t.array);
```



23	24	25	26	27
t object (general info, such as length of following variable addresses)	array var = 000025	array[0] = 1	array[1] = 2	array[2] = 3

Method Argument Passing

```
Test.changeIntArray(t.array);
static void changeIntArray(int[] anIntArray) {
anIntArray = new int[] {100,200,300};}
```



Test t

int[] t.array

anintArray = new int[] {100, 200, 300};

(Memory address reference is overwritten.)

23	24	25	26	27	28	29	30	31
t object (general info, such as length of following variable addresses)	array var = 000025	array[0] = 2	array[1] = 2	array[2] = 3	anintArray var = 000029	anintArray[0] = 100	anintArray[1] = 200	anintArray[2] = 300

Summary

```
class CLASSNAME {
// FINAL CLASS VARIABLE DEFINITIONS
// CLASS VARIABLE DEFINITIONS
// FINAL INSTANCE VARIABLE DEFINITIONS
// INSTANCE VARIABLE DEFINITIONS
// CONSTRUCTOR METHOD DEFINITIONS
// INSTANCE METHOD DEFINITIONS
// CLASS METHOD DEFINITIONS
// MAIN METHOD DEFINITION (OPTIONAL)
}
```

Java SE Built-In Classes

- Organized into packages
  - Groups of related classes
- Examples
  - `java.lang`
  - `java.io; java.nio`
  - `java.math`
  - `java.net; java.rmi; org.omg.CORBA`
  - `java.awt; javax.swing`
  - `java.util`

Java SE Built-In Classes

```java
class MathTester {
public static void main(String[] args) {
double num1 = 2.34;
double num2 = 1.56;
System.out.println(Math.max(num1, num2));
System.out.println(Math.min(num1, num2));
System.out.println(Math.sqrt(num1));
System.out.println(Math.pow(num1, num2));
}
}
```

Java SE Built-In Classes

```
class StringTester {
public static void main(String[] args) {
String string = "";
long startTime1 = System.currentTimeMillis();
for (int i = 0; i < 100000; i++) {
string += "a";
}
long endTime1 = System.currentTimeMillis();
StringBuilder stringBuilder = new StringBuilder();
long startTime2 = System.currentTimeMillis();
for (int i = 0; i < 100000; i++) {
stringBuilder.append("b");
}
long endTime2 = System.currentTimeMillis();
System.out.println("String took: "+(endTime1-startTime1)+"ms");
System.out.println("StringBuilder took: "+(endTime2-startTime2)+"ms");
}}
```

Java SE Built-In Classes

```
class StringTester {
public static void main(String[] args) {
String string = "";
long startTime1 = System.currentTimeMillis();
for (int i = 0; i < 100000; i++) {
string += "a";
}
long endTime1 = System.currentTimeMillis();
StringBuilder stringBuilder = new StringBuilder();
long startTime2 = System.currentTimeMillis();
for (int i = 0; i < 100000; i++) {
stringBuilder.append("b");
}
long endTime2 = System.currentTimeMillis();
System.out.println("String took: "+(endTime1-startTime1)+"ms");
System.out.println("StringBuilder took: "+(endTime2-startTime2)+"ms");
}}
```

Output:

```
String took: 3927ms
StringBuilder took: 0ms
```

Java SE Built-In Classes

```
import java.util.HashSet;

public class HashSetTester {

public static void main(String[] args) {
HashSet<String> mySet = new HashSet<String>();
mySet.add("A");
mySet.add("B");
mySet.add("C");
mySet.add("A");
System.out.println(mySet);
mySet.remove("B");
System.out.println(mySet);
mySet.add("D");
System.out.println(mySet);
}
}
```

Output:

```
[A, B, C]
[A, C]
[D, A, C]
```

Java SE Built-In Classes

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;

class CollectionsTester {

public static void main(String[] args) {
ArrayList<String> listOfStrings = new
ArrayList<String>();
listOfStrings.add("first item");
listOfStrings.add("second item");
listOfStrings.add("third item");
listOfStrings.remove(0); // Remove the first item
System.out.println(listOfStrings);

HashSet<Integer> setOfIntegers = new
HashSet<Integer>();
setOfIntegers.add(2);
setOfIntegers.add(4);
setOfIntegers.add(2);
setOfIntegers.remove(2);
System.out.println(setOfIntegers);

HashMap<String,Integer>
mapOfStringToInteger = new
HashMap<String,Integer>();
mapOfStringToInteger.put("Alice", 4);
mapOfStringToInteger.put("Bob", 3);
mapOfStringToInteger.remove("Alice");
System.out.println(mapOfStringToInteger);
}
}
```

Output:

```
[second item, third item]
[4]
{Bob=3}
```

Conclusions

- Classes and Objects in Java
- Storing Data: Variables
- Defining Behavior: Methods
- Java SE Built-in Classes