Java Basic

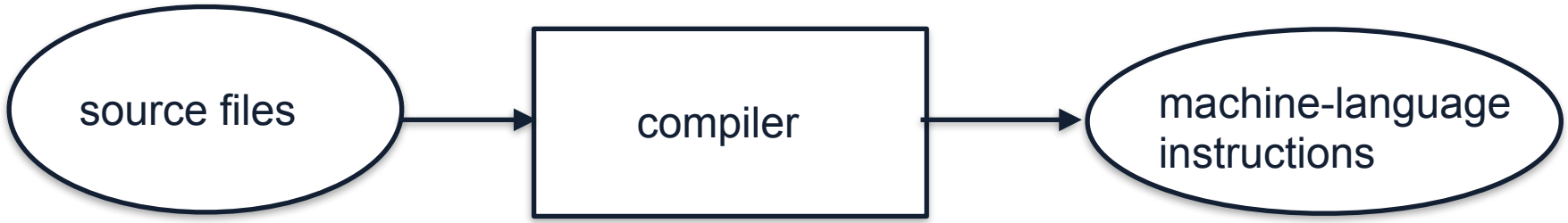
CS 284 C
Instructor: Susan Liu
xueqing.liu@stevens.edu

Learning Objectives

- Java basic:
  - Java environment (JVM) and classes
  - Primitive data types and reference variables
  - the Math class
  - String class
  - Wrapper class for primitive types
  - Defining your own class
  - Array
  - Java I/O

Java Virtual Machine (JVM)

- Introduced in 1995 by Sun company
- Write once, run anywhere (WORA)

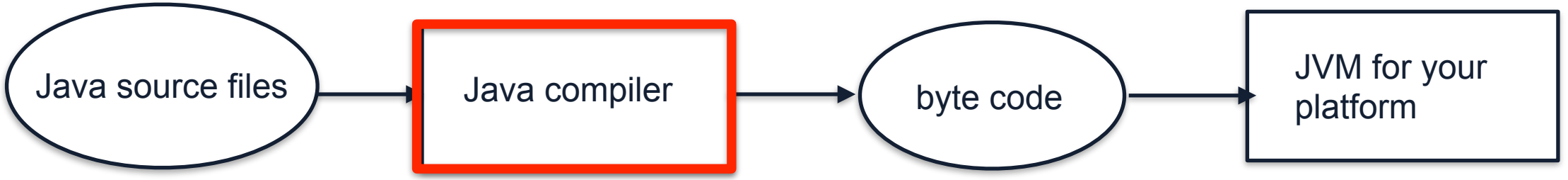


platform dependent, e.g., assembly

(traditional way)

Java Virtual Machine (JVM)

- Introduced in 1995 by Sun company
- Write once, run anywhere (WORA)



(platform **independent**)

(platform **dependent**, e.g., Windows, Linux, Mac)

A: dependent

Java Classes

- A class is a description of a group of entities (objects) that share the same characteristics

```
public class Person {
  // Data Fields
  /** The given name */
  private String givenName = "Mary";
  /** The age*/
  private int age = 30;
}
```

person 1: Mary, age = 30
person 2: Susan, age = 53
…

class

objects

Java Method

- A method is a collection of statements that provide some tasks and return the result

```
public class Person {
  /** getting the age of a person */
  public int getAge(int birthYear){
      return 2020 - birthYear;
  }
}
```

```
int age = getAge(1990);
System.out.println(age);
```

Output: 30

Data Fields and Types

- Variables must be declared with a type before use (unlike Python)

```
private String givenName = “Mary”; // Java
```

```
givenName = “Mary”; #Python
```

- Primitive types (numbers, characters) vs. objects types
- 8 primitive types

byte	-128 to 127
short	-32,768 to 32,767
int	-2,147,483,648 to 2,147,483,647
long	-2^{63} to $2^{63}-1$
float	32-bit IEEE 754 floating point
double	64-bit IEEE 754 floating point
char	Unicode character set
boolean	true, false

Type Compatibility and Conversion

- Widening conversion:
  - int -> double 
  - double -> int 

```
int item = 42;
double realItem = item; // valid

double y = 3.14;
int x = y;
“Compile-time Error: Type mismatch: cannot
convert from double to int”
```

Java Constructor Method

- The constructor method initializes the values of an object

```
public class Person {
  public Person(String givenName, String ID, int age)
{
    …..
}
  public Person(int age){
    …..
}
}
```


Constructor methods have no return type

```
Person mary = new Person("Mary", '123', 23);

Person susan = new Person("Susan", '456', 53);

Person susan = new Person(23);

Person susan = new Person();
```

The `main` Method

- The point where execution begins

```
public class Person {
  public Person(String givenName, String ID, int age) {
    …..
}
  public static void main(String[] args){
     Person mary = new Person(“Mary”, ‘123’, 23);
     …..
}
}
```

Modifying/Getting Values of Objects

- Use the set and get method to modify/get the values of an object

```
public class Person {
  private int age;
  public void setAge(int age) {
      this.age = age;
}
  public String getAge(){
      return this.age;
}
}
```

this refers to the current object

```
 public static void main(String[] args){
Person mary = Person();
mary.setAge(23);
System.out.println(mary.getAge());
}
```

```
 public static void main(String[] args){
Person mary = Person();
mary.age = 23;  ??
System.out.println(mary.age); ??
}
```

Testing Java Methods

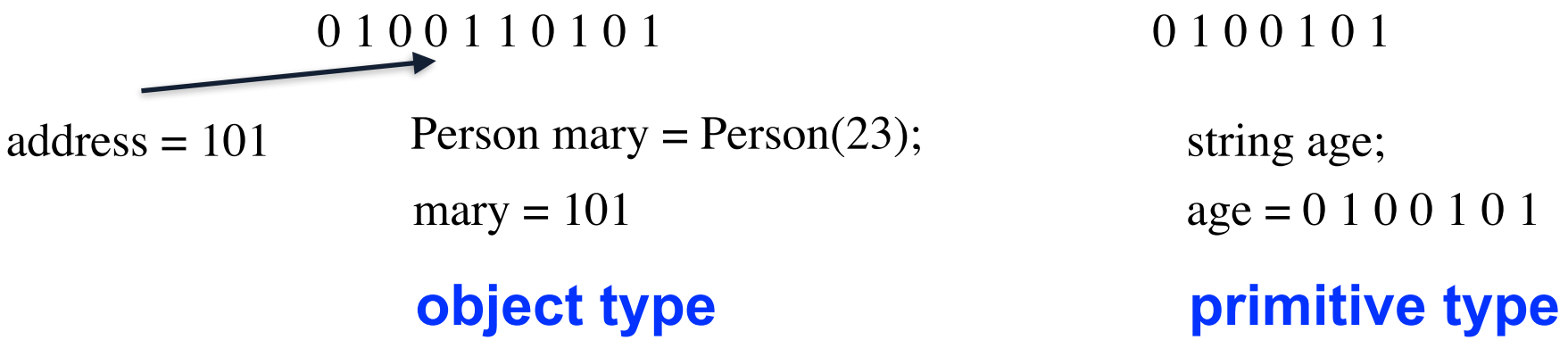
```
public class TestPerson {
 public static void main(String[] args) {
   Person mary = new Person("Mary","123", 30);
   Person susan = new Person("Susan", "456", 53);

   System.out.println("Age of  Mary is " + mary.getAge());
   // prints: Age of Mary is 30

   mary.setAge(35);

   System.out.println("Age of  Mary is " + mary.getAge());
  // prints: Age of Mary is 35
}
```

Referencing Objects



- The Person object Mary is now referenced by the variable `mary`
- `mary` stores the address in memory where the specific object Mary is stored
- Primitive types store the **values** instead of addresses
- Demo 1: Person.java

Static Variable

```
public static int age_static = 30;
```

- Static variables are class variables
  - Shared across all instances
  - Allocated only 1 time

- Instance variables
  - Belong to a specific object
  - Allocated once every object is created

- Demo: Person_2.java

Static Method

- Methods that can be called before any objects being constructed

```
public class Car {
  public void setMileage(int mileage) {
      this.mileage = mileage;
}
  public static void convertMpgToKpl(int Mpg){
      ……
}
}
```

The `Math` Class

- Collection of useful math operations
- All static

Method	Behavior
`static` *numeric* `abs(`*numeric*`)`	Returns the absolute value of its *numeric* argument (the result type is the same as the argument type)
`static double ceil(double)`	Returns the smallest whole number that is not less than its argument
`static double cos(double)`	Returns the trigonometric cosine of its argument (an angle in radians)
`static double exp(double)`	Returns the exponential number *e* (i.e., 2.718 . . .) raised to the power of its argument
`static double floor(double)`	Returns the largest whole number that is not greater than its argument
`static double log(double)`	Returns the natural logarithm of its argument

Recitation Week 1

- Install Eclipse, test code from class
- Joshua: RE, Bhagyesh: RF

Static Variable

```
public static int age_static = 30;
```

- Static variables are class variables
  - Shared across all instances
  - Allocated only 1 time

- Instance variables
  - Belong to a specific object
  - Allocated once every object is created

- Demo: Person_2.java

Static Variable Naming Convention

- Primitive type static variables are all in capital letters

```
// Constants
/** The age at which a person can vote */
private static final int VOTE_AGE = 18;
/** Age at which person considered senior citizen */
private static final int SENIOR_AGE = 65;
```

Static Method Cannot Call Instance Methods/Variables

- Static method cannot call instance method without first creating an object

```
 public static void incAgeTwice() {
        Person.incAge();
        Person.incAge();
}

public static void incAge() {
        Person.age_static = Person.age_static + 1;
}
```

Referencing Objects

0 1 0 0 1 1 0 1 0 1

address = 101

Person mary = Person(23);

mary = 101

object type

0 1 0 0 1 0 1

string age;

age = 0 1 0 0 1 0 1

primitive type

- Primitive types store the values of variables
- Object types store the addresses of variables
- What happens when variables serve as arguments in a function?

Call-by-Value vs. Call-by-Reference

- Java is call-by-value
  - Primitive type: call-by-value
  - Object type: call-by-reference
- PLs that are call-by-value
  - Java, C#, Python, Ruby, etc.
- PLs that are NOT call-by-value
  - Fortran is call-by-reference
- Demo: Person_3.java Person_4.java

```
public class Person {
  public void incAge(int age) {
     age = age + 1;
}
  public static void main(String[] args){
      Person mary = new Person(23);
      int mary_age = 23;
      incAge(mary_age); // what is mary_age?
  }
}
```

Object-Oriented Programming

- Object-oriented programming is a programming paradigm based on the concept of "objects", which can contain data, in the form of fields, and code, in the form of procedures

- C is not object-oriented
  - C is procedural

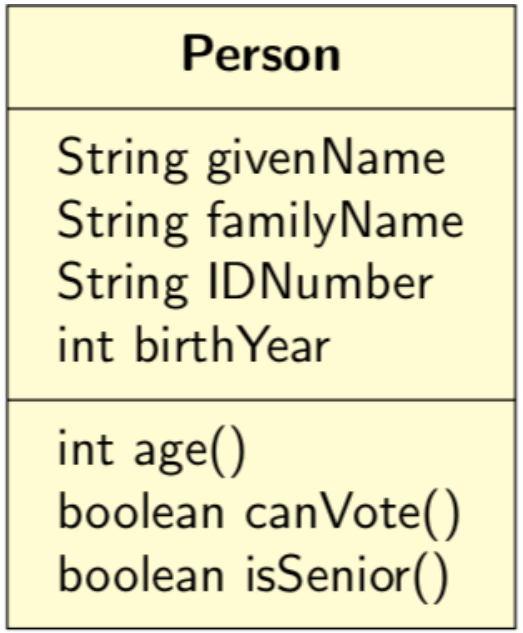
- What is the advantage of object-oriented programming language over procedure-based language?

  Encapsulation; inheritance; polymorphism; abstract

UML Diagrams

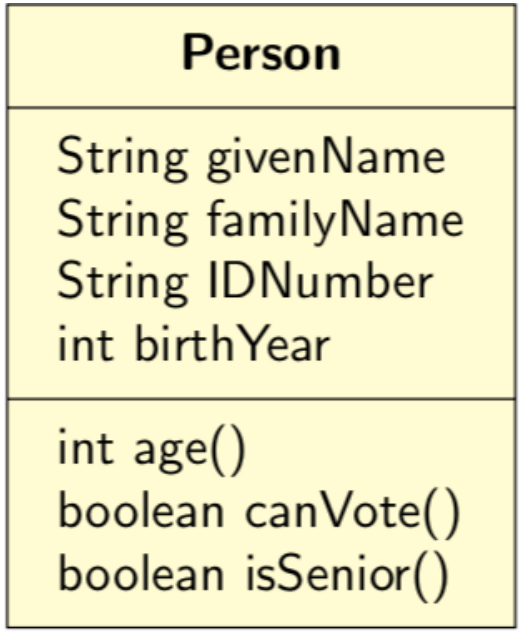
- The unified modeling language (UML) represents the unification of earlier object-oriented design modeling techniques

- Why UML?



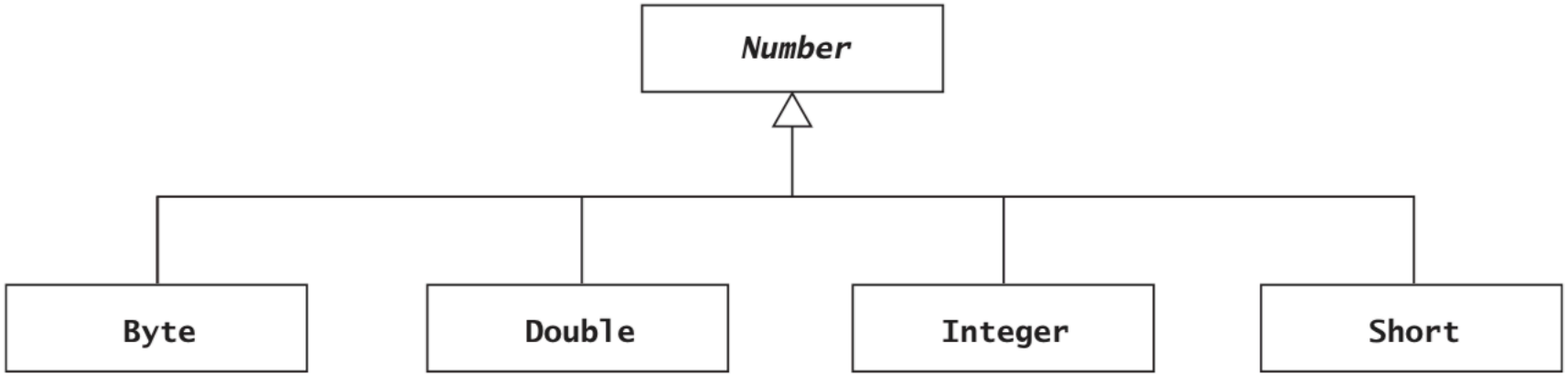
Style of UML diagrams

- The classes are represented by rectangles
- Lines between classes represent the relationships between classes
- Use of camel case notations such as `givenName`



UML Diagrams Show Essential Information

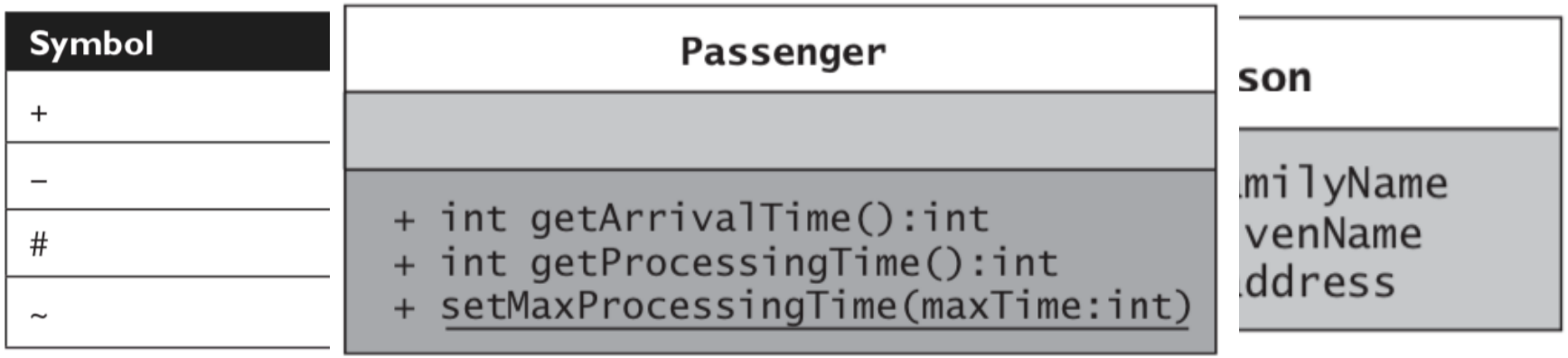
- A class carries a lot of information
  - If all the information is included in the UML diagram, the diagram will look cluttered
  - In practice, we show only the essential information



UML diagrams

- Interface indicator: use double angle brackets to indicate the class is an interface
  - e.g., {abstract}

- Visibility indicators, static attributes, and parameter name and types



Arrays

- In Java, Array is an object
- Different ways to declare array and allocate its storage

```
int[] scores = new int[5];
```

```
int[] scores = {1, 2, 3};
```

```
int[] scores;
scores = {1,2,3};
```

- Array of user-defined type

```
Person[] people;
int n     = 3+4;
people    = new Person[n];
people[0] = new Person("Elliot","Koffman","123",1942);
```

Arrays are Initialized by 0

```
int[] scores   = new int[5];
for (int i=0; i<5; i++) {
  System.out.println(scores[i]);
};
```

Output:

0
0
0
0
0

```
String[] scores   = new String[5];
for (int i=0; i<5; i++) {
  System.out.println(scores[i]);
};
```

Output: ?

System.arraycopy

- Deep copy an array:

System.arraycopy(*source*, *sourcePos*, *destination*, *desPos*, *numElements*);

```
int[] scores   = new int[5];
int[] double_scores = new int[5];
scores = double_scores;
double_scores[1] = 5;
System.out.println(scores[1]);
```

Output: ?

Alternative Ways of For-loop for Array

- There is an enhanced way of writing for-loop for collections, array included
- Rather than

```
for (int i=0; i<5; i++) {
  System.out.println(scores[i]);
};
```

- We can write

```
for (int i : scores) {
  System.out.println(scores[i]);
};
```

System.arraycopy

- Deep copy an array:

System.arraycopy(*source*, *sourcePos*, *destination*, *desPos*, *numElements*);

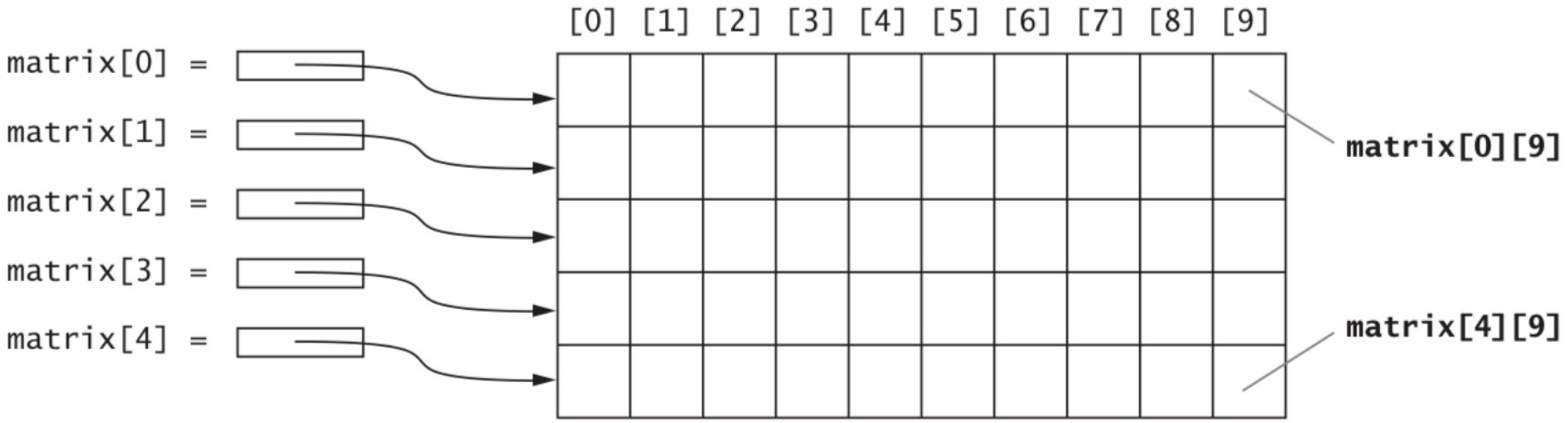
```
int[] scores   = new int[5];
int[] double_scores = new int[5];
scores = double_scores;
double_scores[1] = 5;
System.out.println(scores[1]);
```

Output: ?

Two Dimensional Arrays

- The statement allocates storage for a two dimensional array

```
double[][] matrix = new double[5][10];
```




```
double[][] matrix = new double[5, 10];
```

Java String Operations

- String operations process sequence of characters
- Assume keyboard is a String variable that contains “qwerty”

```
keyboard.charAt(0) // q
keyboard.length()  // 6
keyboard.indexOf('o') // -1
keyboard.indexOf('y') // 5
String upper=keyboard.toUpperCase();
```

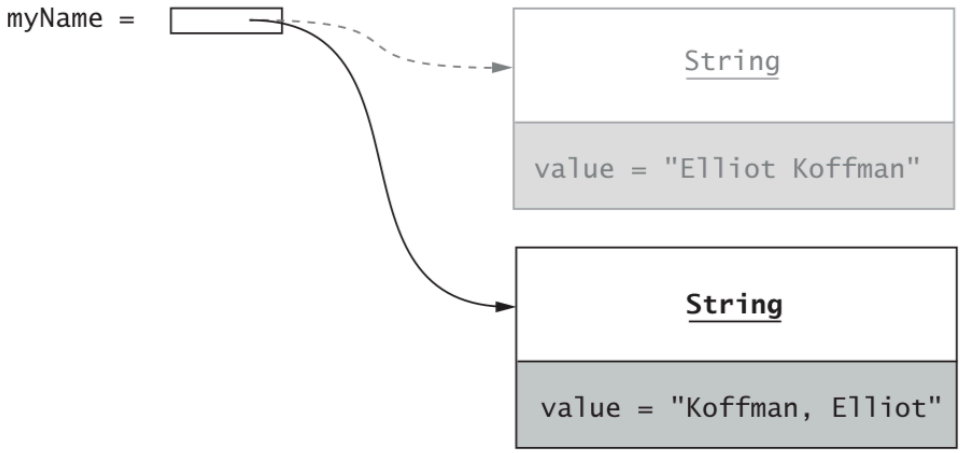
- toUpperCase() does not change the value of keyboard

Strings are Immutable

- You cannot modify a String object:

```
myName[0]= 'X'; // invalid, String is not an Array
myName.charAt(0)= 'X'; // invalid
```

- When modifying a String object, Java will create a new object that contains the modified sequence, the original object still exists



StringBuffer and StringTokenizer

- StringBuffer also stores string objects
  - However, the content can be changed

```
StringBuffer sB3 = new StringBuffer("happy");
sB3.append("birthday to you");
```

- StringTokenizer
  - Turn a sentence into sequence of words

```
String personData = "Doe, John 5/15/65";
StringTokenizer st = new StringTokenizer(personData, ",/");
```

Tokenize a String

- Split a list of numbers by comma

```
String personData = “12, 3,456, 78”;
String[] newData = personData.split(",", -1);
System.out.println(newData.length);
```

- Split a list of numbers by regular expression

```
String personData = “12, 3,456, 78”;
String[] newData = personData.split(“,  “, -1);
System.out.println(newData.length);
```

Java Method toString

- The toString method creates a string object that represents the information stored in an object

```
public String toString() {
   return "Given name: " + givenName + "\n"
+ "Family name: " + familyName + "\n"
+ "ID number: " + IDNumber + "\n"
+ "Year of birth: " + birthYear + "\n";}
```

- Automatically apply toString:

```
System.out.println(person.toString());
System.out.println(person);
```

Java Method equals

```
/** Compares two Person objects for equality.
    @param per The second Person object
    @return true if the Person objects have same
            ID number; false if they don't
*/

 public boolean equals(Person per) {
  if (per == null)
    return false;
  else
    return IDNumber.equals(per.getIDNumber());
 }}
```

```
Person mary = new
Person('abc');
susan = new Person('abc');
System.out.println(mary ==
susan);
```

Wrapper Class for Primitive Types

- Primitive numeric types are not objects, but sometimes they need to be processed like objects
- e.g., When primitive types must be inserted into collections
- Java provides wrapper classes whose objects contain primitive-type value

byte	Byte	float	Float
boolean	Boolean	int	Integer
char	Character	long	Long
double	Double	short	Short

- “wrap” and “unwrap” an object automatically

```
Integer a = new Integer(1);
int b = a;
```

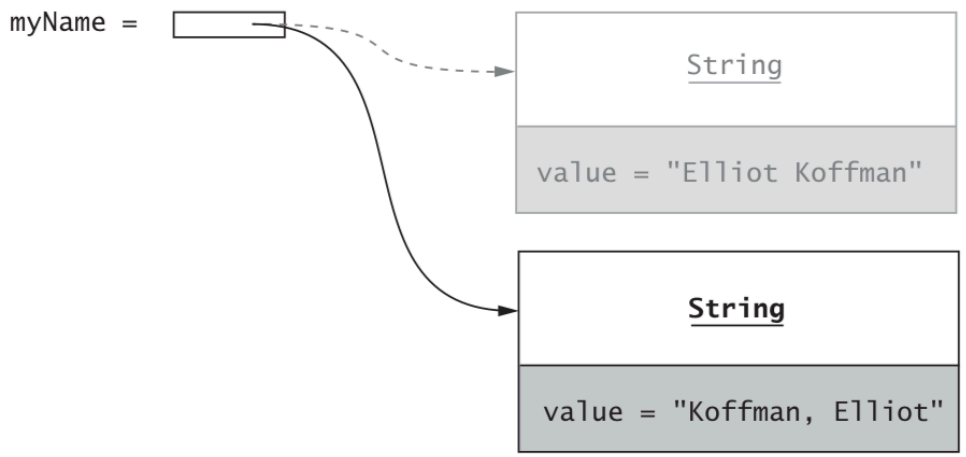
Java Comments

```
public class Person {
  // Data Fields
  /** The given name */
  private String givenName;
  /** The family name */
  private String familyName;
  /** The ID number */
  private String IDNumber;
  /** The birth year */
  private int birthYear = 1900;
```

//vs/** ... */vs/* ... */

Garbage Collection

- Storage space for objects no longer referenced is automatically reclaimed by Java garbage collector



- C and C++ do not have a garbage collection, programmers have to delete the objects they create

Programming Style

- Some programmers unnecessarily write if statements to return a boolean value:

```
return IDNumber.equals(per.IDNumber);
```

- They write

```
if (IDNumber.equals(per.IDNumber))
        return true;
else
        return false;
```