

# More sophisticated behavior

Using library classes to implement  
some more advanced functionality

# The Java class library

- Thousands of classes
- Tens of thousands of methods
- Many useful classes that make life much easier
- A competent Java programmer must be able to work with the libraries.

# Working with the library

You should:

- know some important classes by name;
- know how to find out about other classes.

Remember:

- We only need to know the interface, not the implementation.

# Example

```
String str = "Some example string";  
if (str.startsWith("something")) {  
    // do something ...  
}
```

- Where does ‘startsWith’ come from?
- What is it? What does it do?
- How can we find out?

# Reading class documentation

- Documentation of the Java libraries in HTML format;
- Readable in a web browser
- Class **API**:  
***Application Programming Interface***
- Interface description for all library classes

# Interface vs implementation

*The documentation includes*

- the name of the class;
- a general description of the class;
- a list of constructors and methods
- return values and parameters for constructors and methods
- a description of the purpose of each constructor and method



the *interface* of the class

# Interface vs implementation

*The documentation **does not** include*

- private fields (most fields are private)
- private methods
- the bodies (source code) for each method



the *implementation* of the class



# Using library classes

- Classes from the library must be imported using an *import* statement (except classes from *java.lang*).
- They can then be used like classes from the current project.



# Packages and import

- Classes are organised in packages.
- Single classes may be imported:

```
import java.util.ArrayList;
```

- Whole packages can be imported:

```
import java.util.*;
```

# Information Hiding

- The principle of Information Hiding states that **internal details** of a class's implementation should be hidden from other classes.
- It ensures better modularization of an application.

# Information hiding

- Data belonging to one object is hidden from other objects.
- Know what an object can do, not how it does it.
- Information hiding increases the level of *independence*.
- Independence of modules is important for large systems and maintenance.

# public vs private

- **Public** members (fields, constructors, methods) are accessible to all other classes.
- **Private** members are accessible only within the same class.

# default / package access

- Not specifying any access modifier means “default access”, or “package-private”.
- Package access members are accessible to any class within the same package.

# Which access modifier ?

- According to the principle of “Information Hiding”, programmers should use the most restrictive access modifier possible.
- Simply, prefer “private” over “public” whenever possible.
- Generally:
  - Almost all fields must be **private**.
  - Methods that implement a behaviour of this class must be **public**.
  - Methods with internal usage must be **private**.

# final / constant fields

- Class fields can be declared **constant**, using the “**final**” keyword.
- Final / constant fields can be initialized at the constructor.
- By convention, Java programmer use ALL\_CAPS names for final fields.

```
private final int SIZE = 10;
```



# static / class members

- The “static” keyword is used to specify **class members**.
- Class members don't need an object to be accessed; they are accessible using the class name.
- Values of class members are **shared** among all objects.

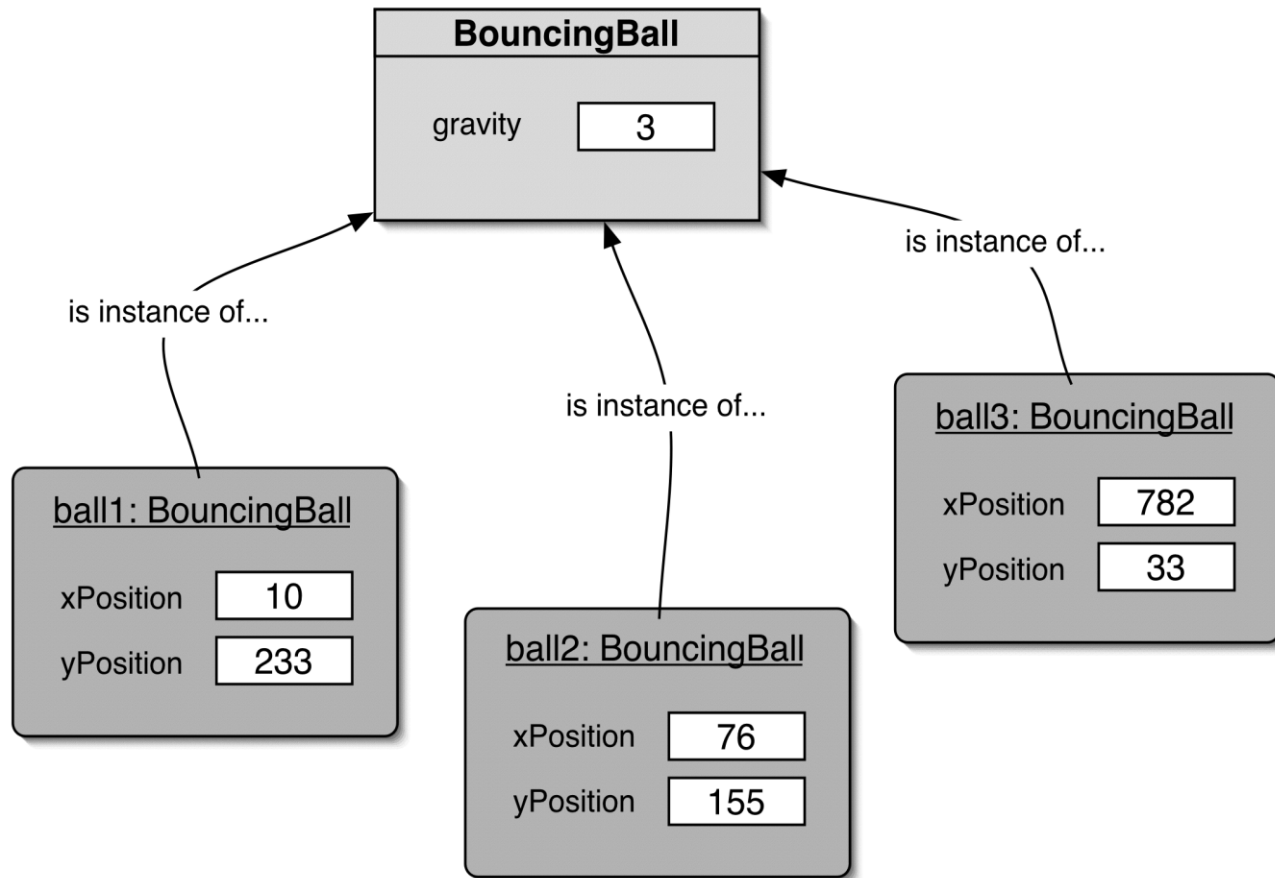
```
public class Test {  
    private static int shared = 0;  
    ...  
}
```

# static + final = Class variable

```
private static final int GRAVITY = 3;
```

- **private**: access modifier, as usual
- **static**: class variable
- **final**: constant

# Class variables



# Immutability

- Immutable objects are objects that once they are created, their state cannot be modified.

```
public class ImmutableClass {  
    private int value;  
    public ImmutableClass(int value) {  
        this.value = value;  
    }  
    public int getValue() {  
        return value;  
    }  
}
```

# Immutability

- A well-know immutable class in Java is the “String” class.

```
String str = "testing";  
str.toUpperCase();  
System.out.println(str); // prints:  testing
```

```
str = str.toUpperCase();  
System.out.println(str); // prints:  TESTING
```

# Side note: String equality

```
if (input == "bye") {  
    ...  
}
```

tests identity

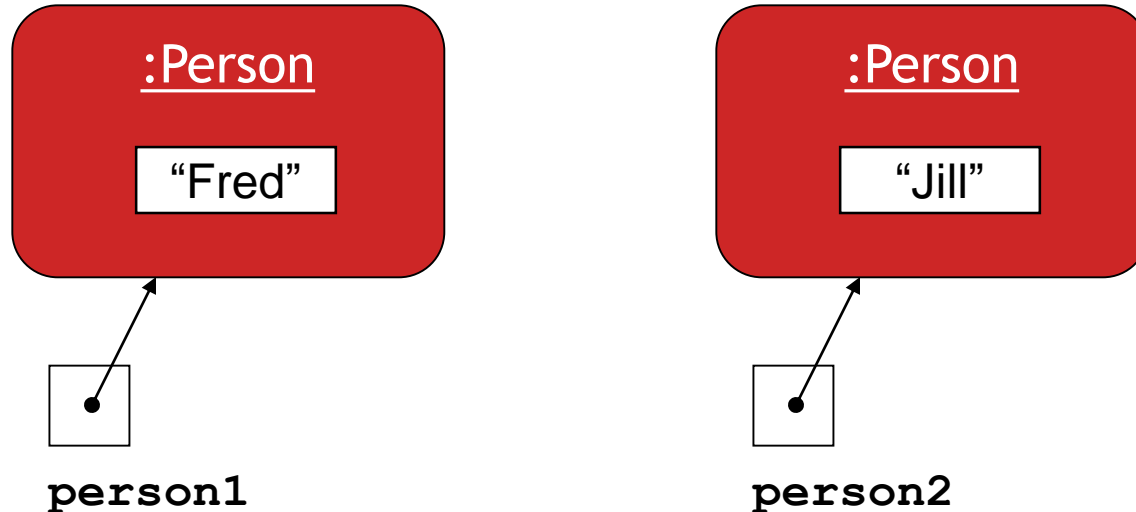
```
if (input.equals("bye")) {  
    ...  
}
```

tests equality

- Strings should always be compared with `.equals`

# Identity vs equality

Other (non-String) objects:

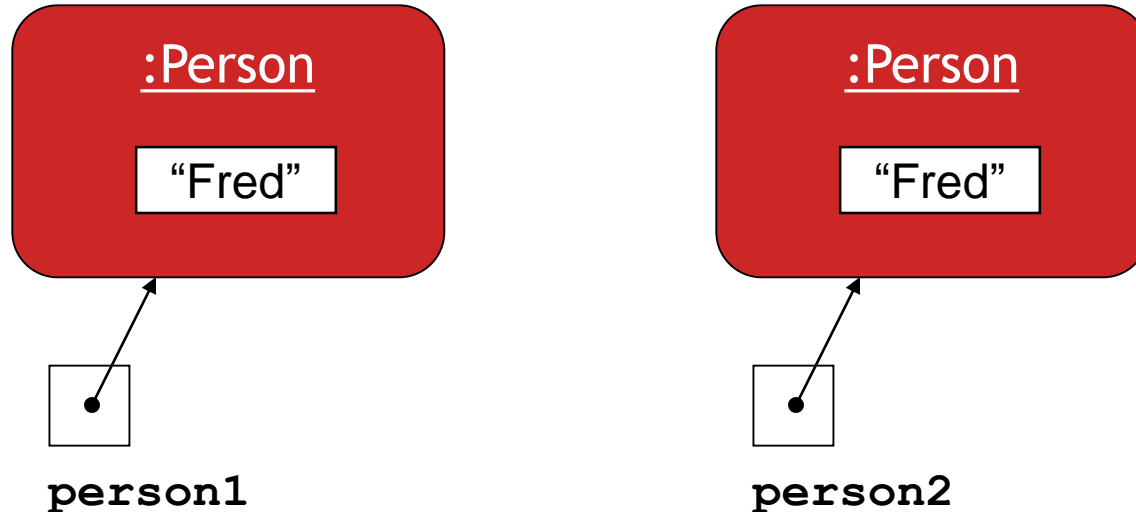


`person1 == person2 ?`



# Identity vs equality

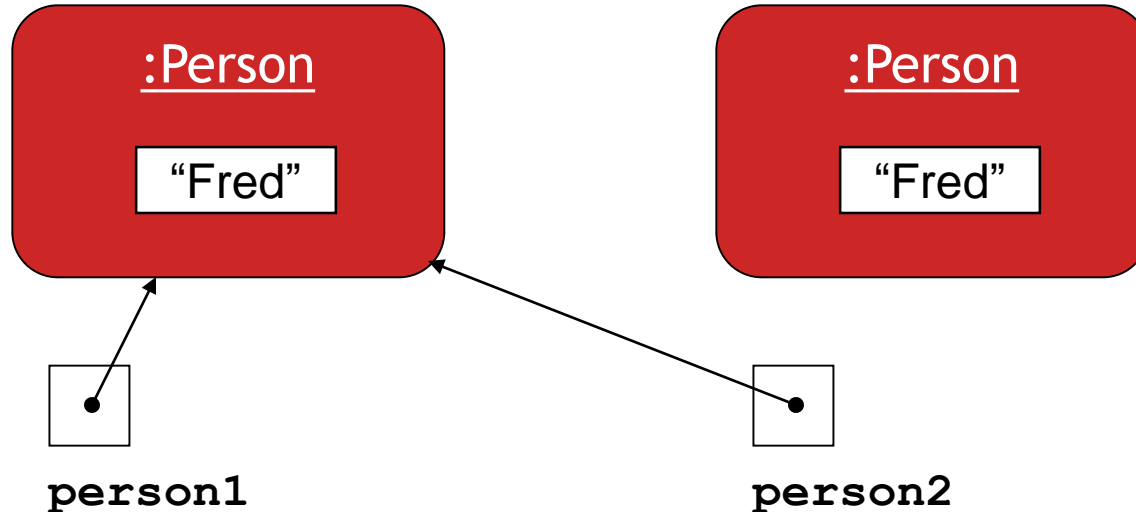
Other (non-String) objects:



`person1 == person2 ?`

# Identity vs equality

Other (non-String) objects:

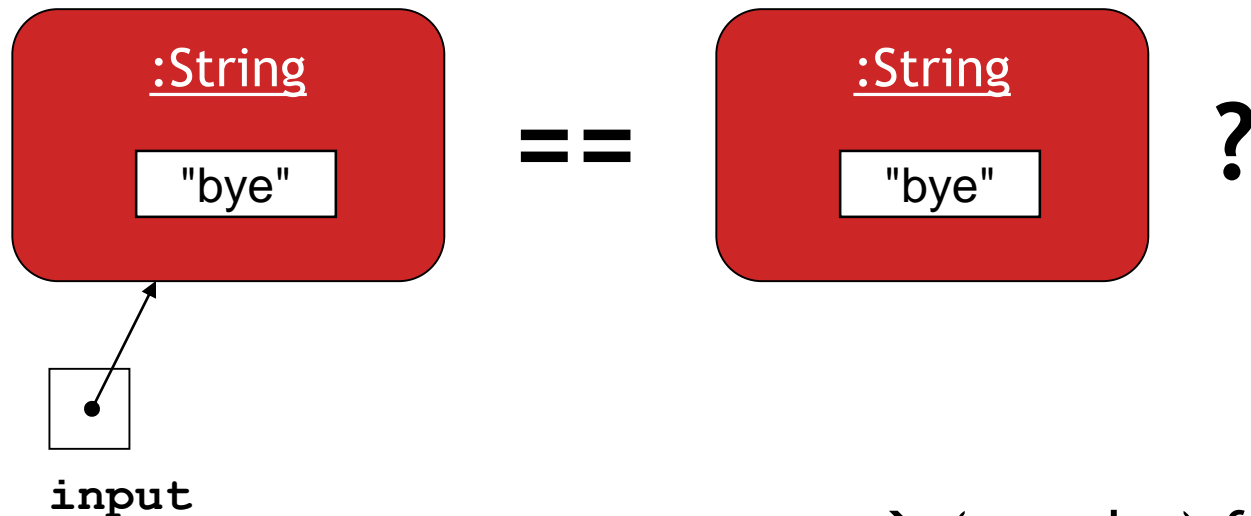


`person1 == person2 ?`

# Identity vs equality (Strings)

```
String input = reader.getInput();  
if (input == "bye") {  
    ...  
}
```

**== tests identity**

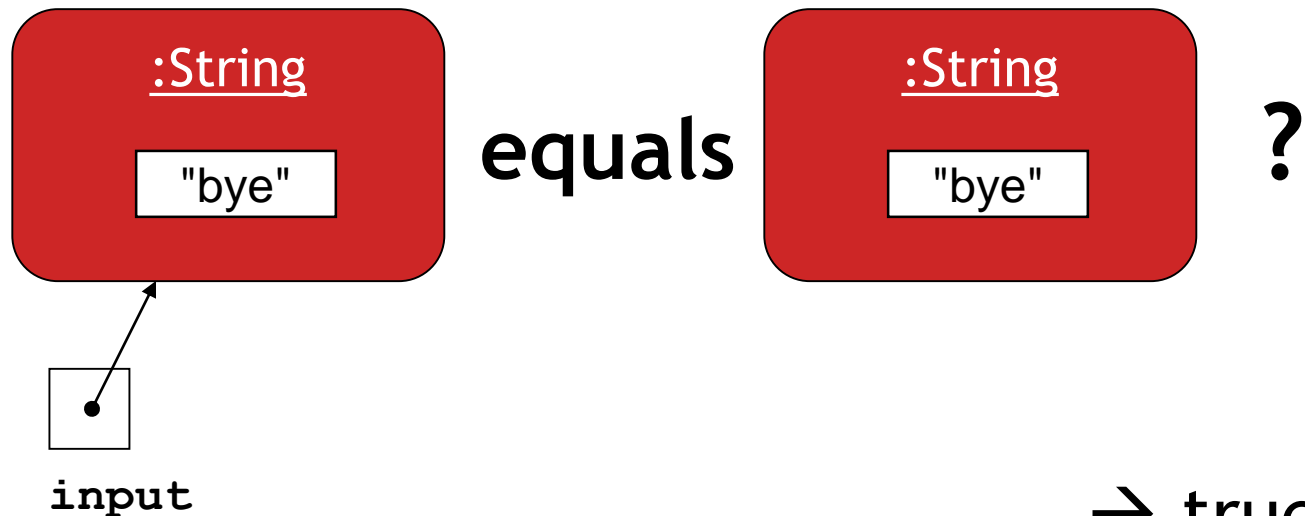


→ (may be) false!

# Identity vs equality (Strings)

```
String input = reader.getInput();  
if (input.equals("bye")) {  
    ...  
}
```

**equals tests  
equality**



**→ true!**

# Using Random

- The library class Random can be used to generate random numbers

```
import java.util.Random;  
...  
Random randomGenerator = new Random();  
...  
int index1 = randomGenerator.nextInt();  
int index2 = randomGenerator.nextInt(100);
```

# Maps

- Maps are collections that contain pairs of values.
- Pairs consist of a key and a value.
- Lookup works by supplying a key, and retrieving a value.
- An example: a telephone book.

# Using maps

- A map with Strings as keys and values

:HashMap

"Charles Nguyen"

"(531) 9392 4587"

"Lisa Jones"

"(402) 4536 4674"

"William H. Smith"

"(998) 5488 0123"



# Using maps

```
HashMap<String, String> phoneBook =  
    new HashMap<String, String>();  
  
phoneBook.put("Charles Nguyen", "(531) 9392 4587");  
phoneBook.put("Lisa Jones", "(402) 4536 4674");  
phoneBook.put("William H. Smith", "(998) 5488 0123");  
  
String phoneNumber = phoneBook.get("Lisa Jones");  
System.out.println(phoneNumber);
```

# Using sets

```
import java.util.HashSet;
import java.util.Iterator;
...
HashSet<String> mySet = new HashSet<String>();
```

```
mySet.add("one");
mySet.add("two");
mySet.add("three");
mySet.add("one");
```

```
Iterator<String> it = mySet.iterator();
while(it.hasNext()) {
    call it.next() to get the next object
    do something with that object
}
```

Compare this  
to ArrayList  
code!

# Review

- Java has an extensive class library.
- A good programmer must be familiar with the library.
- The documentation tells us what we need to know to use a class (interface).
- The implementation is hidden (information hiding).
- We document our classes so that the interface can be read on its own (class comment, method comments).

## شعر امروز

گر دامن وصل تو کشم، جنگی نیست!  
ور طعنه عشقت شنوم، ننگی نیست!

با وصل خوشت، می زنم و می گیرم  
وصلی که در او فراق را رنگی نیست!