Software Practice 1 - OOP (1) – Class and Method

- What is OOP?
- Defining Classes
- Using Classes
- References vs. Values
- Encapsulate

Prof. Joonwon Lee

T.A. Jaehyun Song
Jongseok Kim (42)

T.A. Sujin Oh
Junseong Lee (43)

Objects in real world

- **Represent the real world**

Baby

Objects in real world

- **Represent the real world**

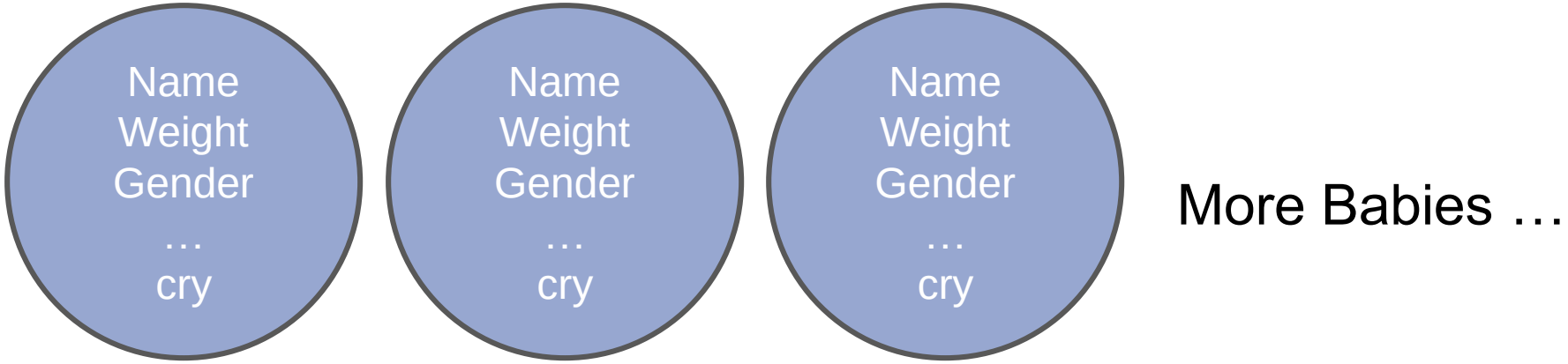
Baby

Name
Gender
Weight
Poops

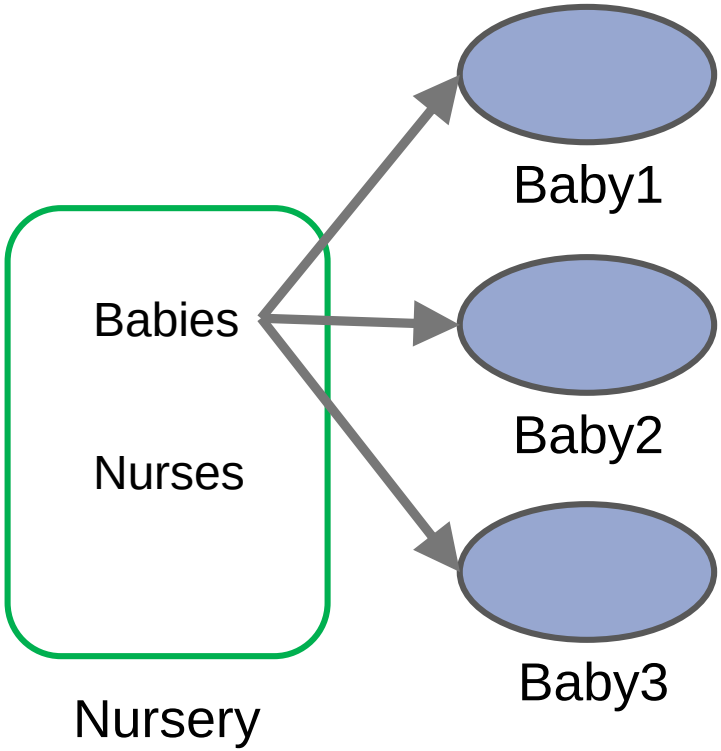
Objects in real world

Baby

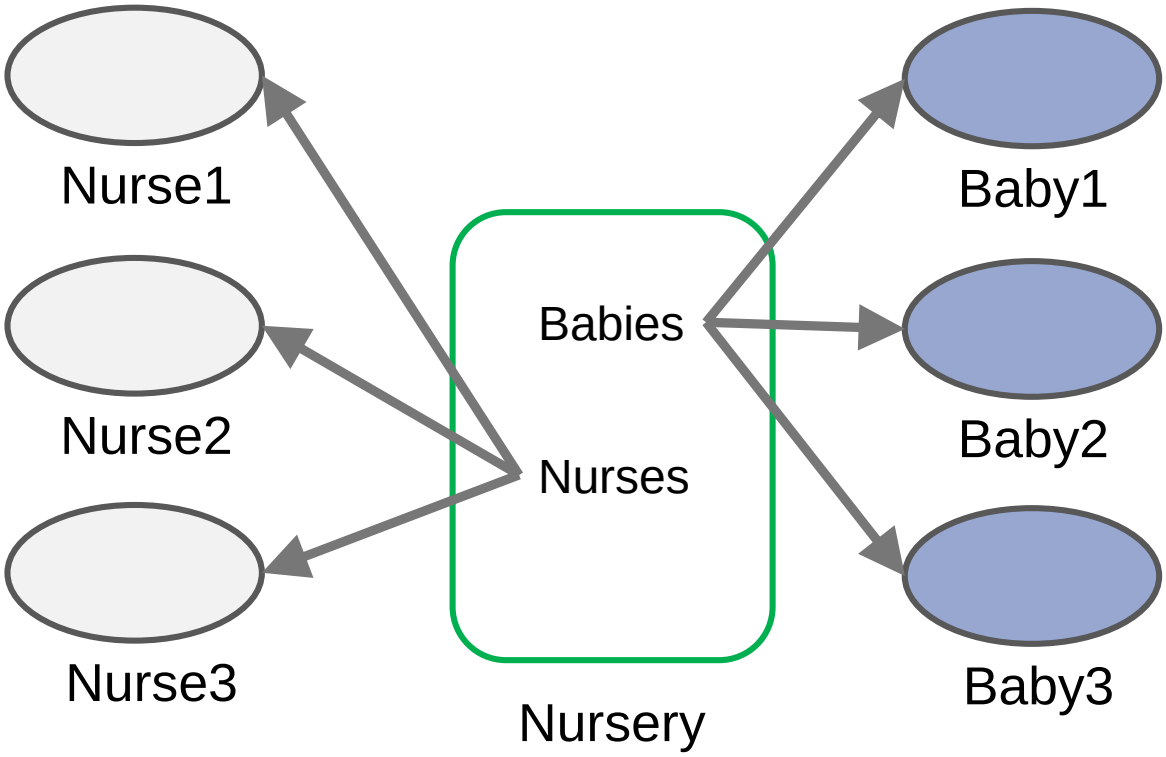
Objects in real world



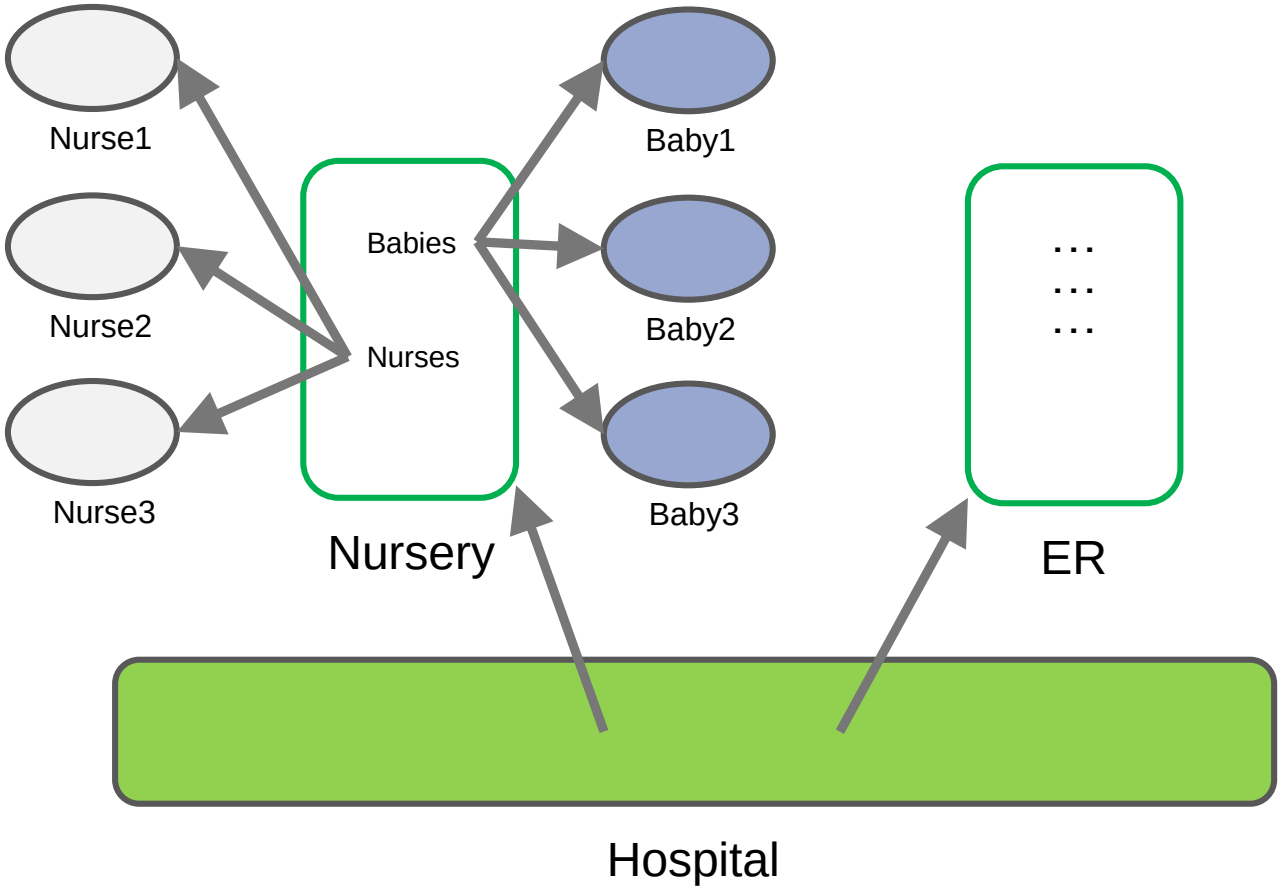
Objects in real world



Objects in real world

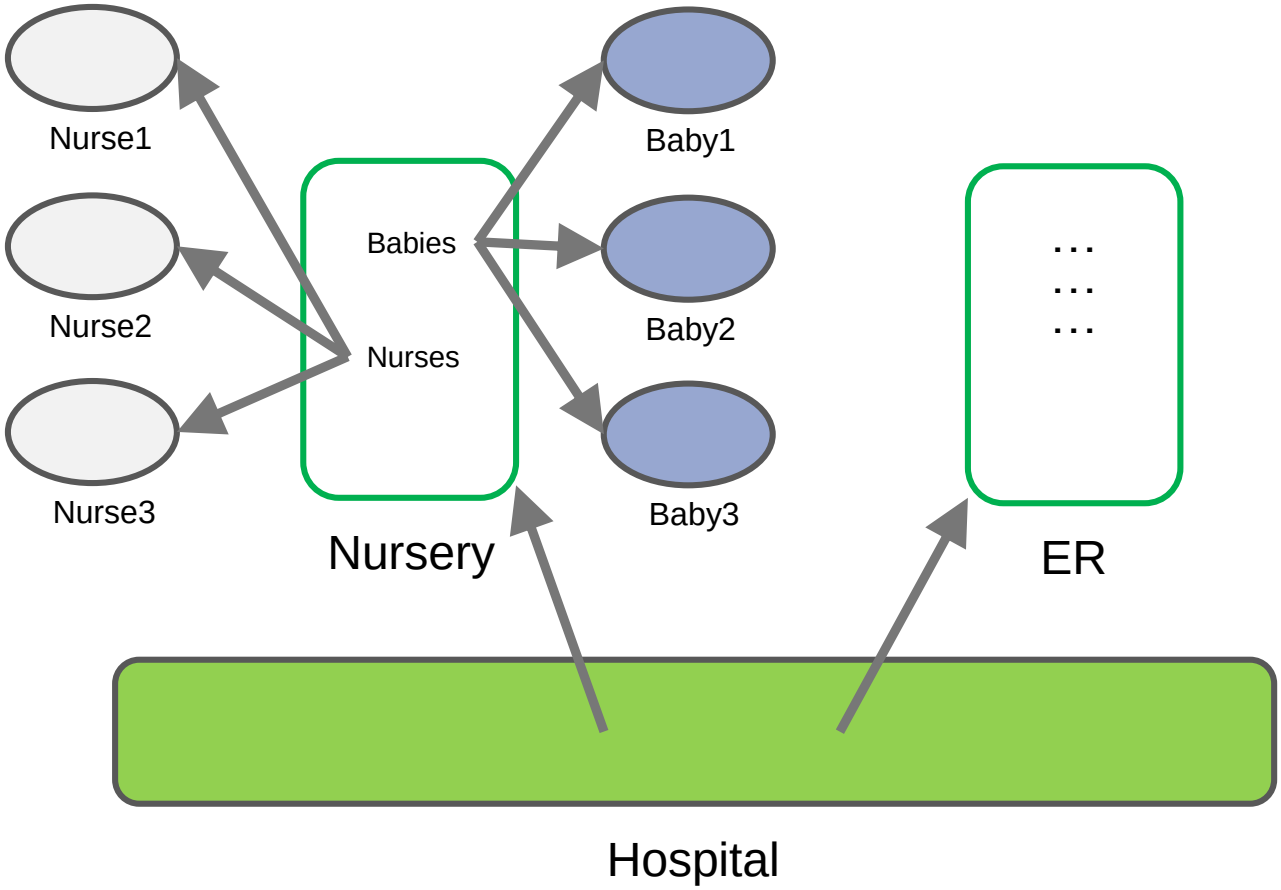


Objects in real world



Objects in real world

- **What do we need to model this objects with programming language?**



Object Oriented Programming

- **Definition**
  - A method of programming based on a hierarchy of classes, and well-defined and cooperating objects.

- **Class** – a structure that defines
  - the **field**s to store the data
  - the **method**s to work on that data

- **Object** – an executable copy of a class

Object Oriented Programming

- **A class**
  - can be inherited by only one other class
  - can implement one or more interfaces

- **An object**
  - can establish the relationship with other objects through the reference
  - encapsulates some fields or methods for hiding them from other objects

DEFINING CLASSES

Class definition

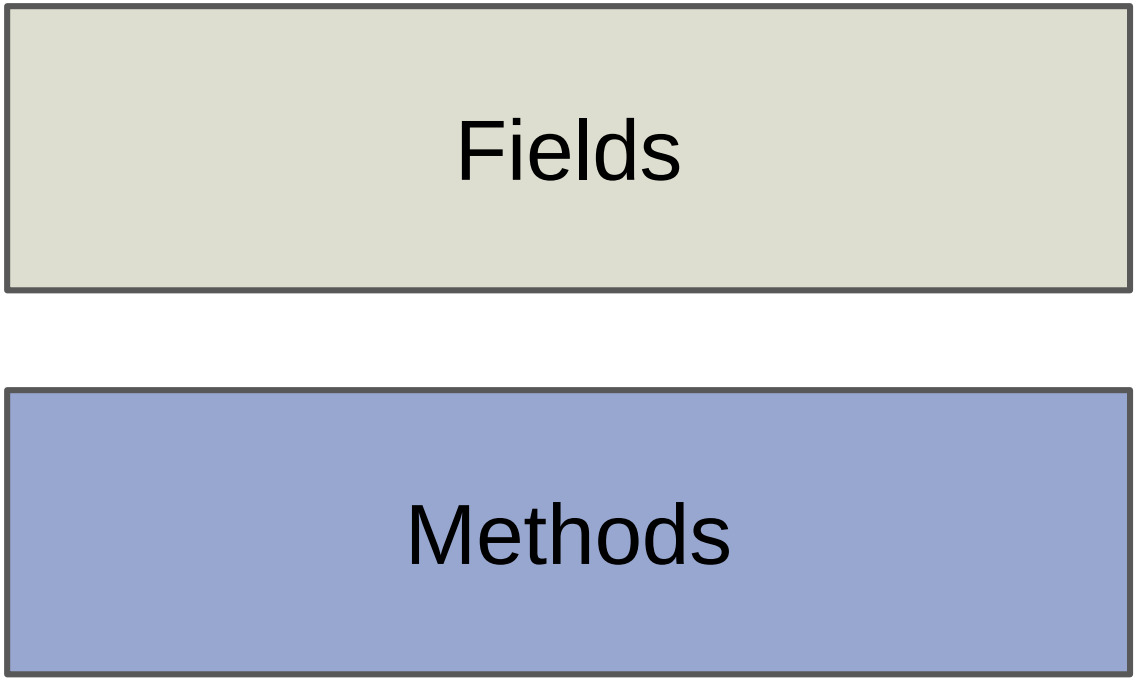
```
public class Baby {
  String name;
  boolean gender;
  double weight;
  int numPoops = 0;

  void poop() {
    numPoops += 1;
    System.out.println (“Dear mother, “ +
    “I have pooped. Ready the diaper.”);
  }
}
```

Let’s declare a baby!

```
public class Baby {

}
```



Baby fields

```
public class Baby {

    TYPE var_name;
        OR
    TYPE var_name = some_value;

}
```

Baby fields

```
public class Baby {
  String name;
  boolean gender;
  double weight = 5.0;
  int numPoops = 0;









}
```

Baby Siblings?

```
public class Baby {
  String name;
  boolean gender;
  double weight = 5.0;
  int numPoops = 0;
  XXXXX YYYYY;




}
```

Baby Siblings?

```
public class Baby {
  String name;
  boolean gender;
  double weight = 5.0;
  int numPoops = 0;
  Baby[] siblings;




}
```

Baby methods

```
public class Baby {
  String name = “Slim Shady”;
  ...
  void sayHi() {
    System.out.println (
    “Hi, my name is.. “ + name);
  }
}
```

Baby methods

```
public class Baby {
  double weight = 5.0;
  ...
  void eat(double foodWeight) {
    if (foodWeight >= 0 &&
        foodWeight < weight) {
      weight += foodWeight;
  }
}
```

Baby class

```
public class Baby {
  String name;
  boolean gender;
  double weight;
  int numPoops = 0;
  Baby[] siblings;

  void sayHi() {...}
  void eat(double foodWeight) {...}
}
```

Ok, let's make this baby!

```
Baby myBaby = new Baby();
```

But what about his/her name? gender?

Constructors

```
public class CLASSNAME {
  CLASSNAME ( ) {
  }

  CLASSNAME ( [ARGUMENTS] ) {
  }

}
```

```
CLASSNAME obj1 = new CLASSNAME();
CLASSNAME obj2 = new CLASSNAME([ARGUMENTS]);
```

Constructors

- **Constructor name == the class name**
- **No return type – never returns anything**
- **Usually initialize fields**
- **All classes need at least one constructor**
  - If you don't write one, defaults to

```
CLASSNAME () {
}
```

Baby constructor

```
public class Baby {
   String name;
   boolean gender;
   Baby(String myName, boolean myGender) {
      name = myName;
      gender = myGender;
   }
}
```

Final Baby class

```
public class Baby {
  String name;
  boolean gender;
  double weight;
  int numPoops = 0;
  Baby[] siblings;

  Baby() {...}
  void sayHi() {...}
  void eat(double foodWeight) {...}
}
```

USING CLASSES

Classes and Instances

```
// class Definition
public class Baby { … }

// class Instances
Baby shiloh = new Baby (“Shilo Jolie-Pitt”, true);
Baby knox   = new Baby (“Knox Jolie-Pitt”,  true);
```

Accessing fields

- **Object.FIELDNAME**

```
Baby shiloh = new Baby (“Shiloh Jolie-Pitt”, true);
System.out.println (shiloh.name);
System.out.println (shiloh.numPoops);
```

Using methods

- **Object.METHODNAME([ARGUMENTS])**

```
Baby shiloh = new Baby (“Shiloh Jolie-Pitt”, true);
shiloh.sayHi();  // “Hi, my name is.. Shiloh Jolie-Pitt”
shiloh.eat(1);
```

REFERENCES VS. VALUES

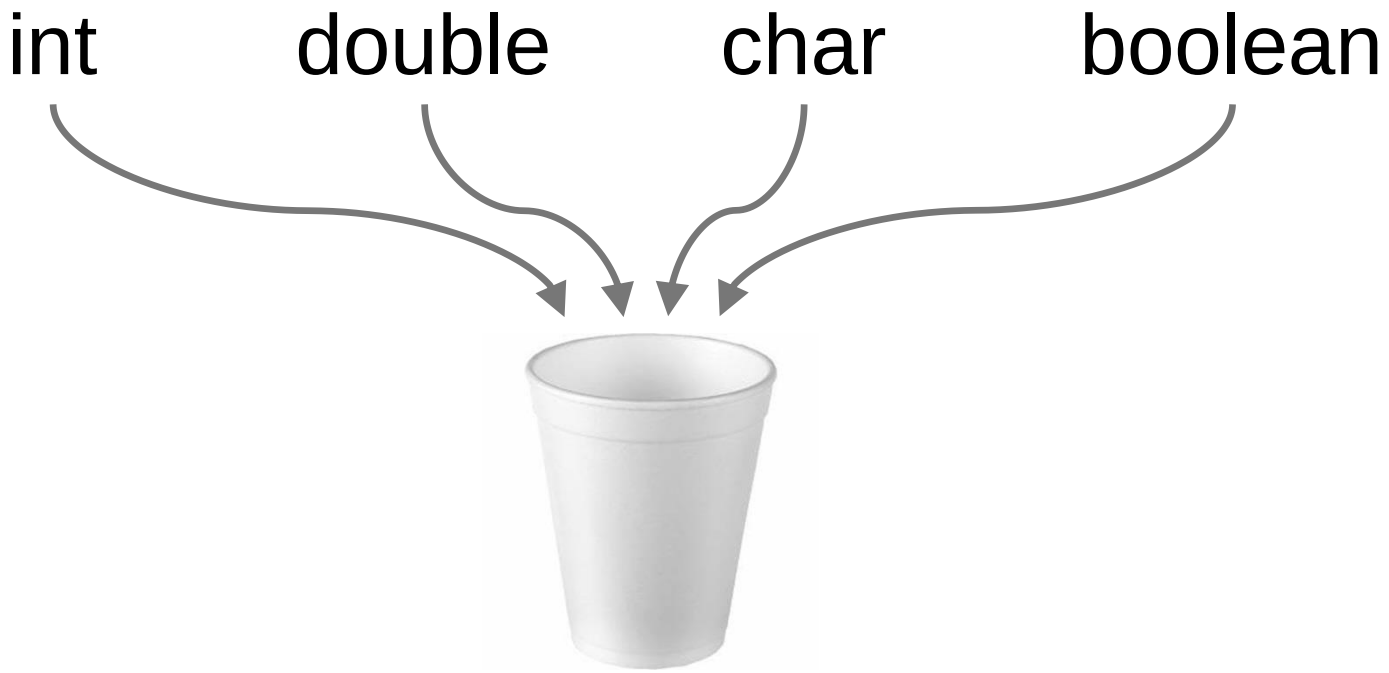
Primitives vs. References

- **Primitive types are basic java types**
  - int, long, double, boolean, char, short, byte, float
  - The actual values are stored in the variable

- **Reference types are arrays and objects**
  - Class, Interface, Array, Enum, ETC
  - Ex) String, int[], Baby, …

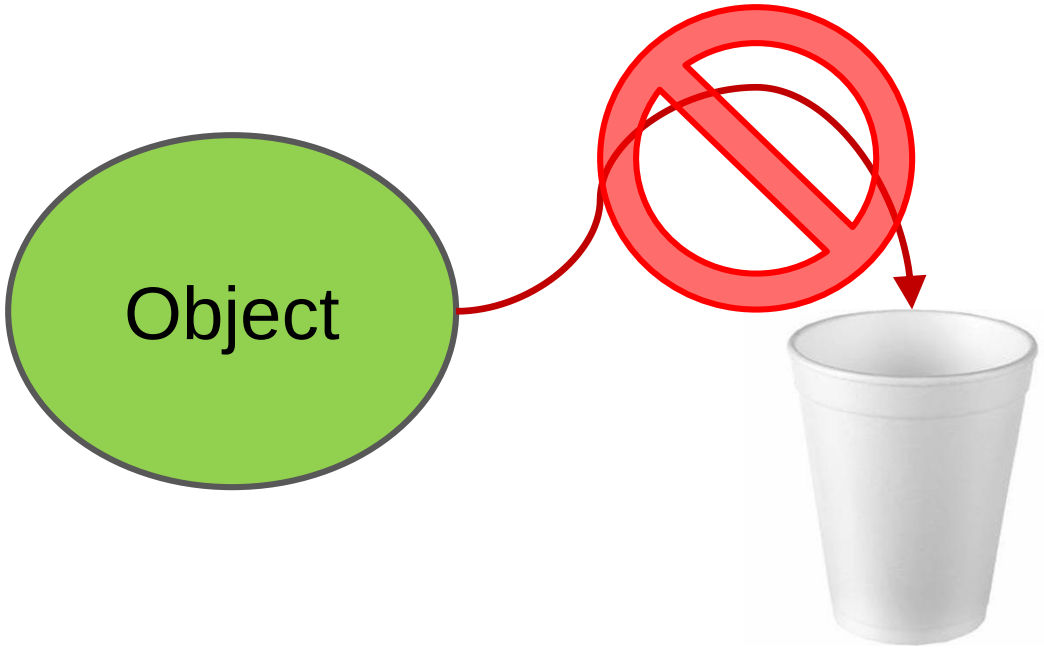
How java stores primitives

- **Variables are like fixed size cups**
- **Primitives are small enough that they just fit into the cup**



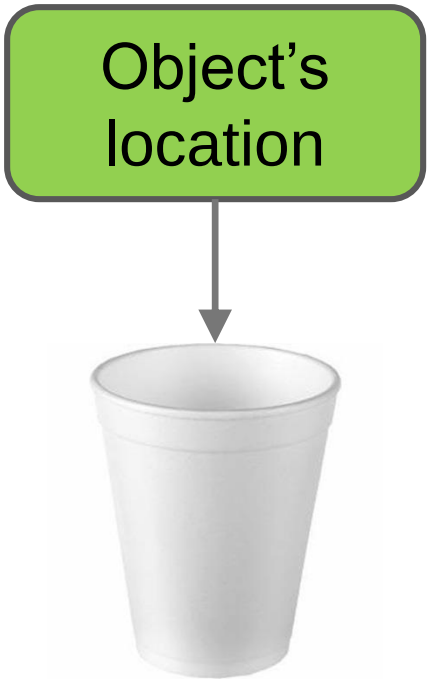
How java stores objects

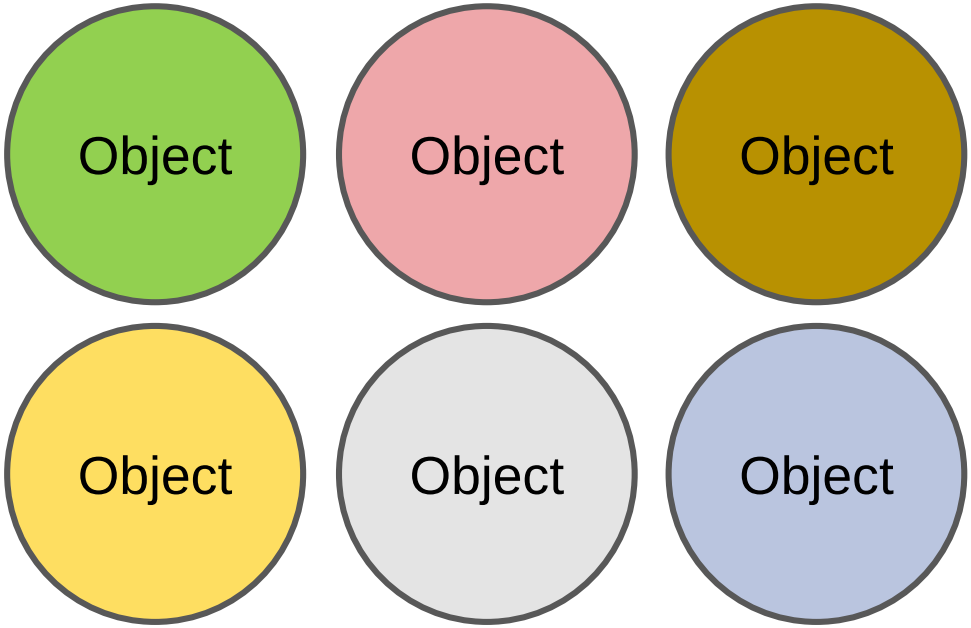
- **Objects are too big to fit in a variable**
  - Stored somewhere else
  - Variable stores an address that locates the object



How java stores objects

- **Objects are too big to fit in a variable**
  - Stored somewhere else
  - Variable stores an address that locates the object





References

- **The object's location is called a reference**
- **== compares the references**

```
Baby shiloh1 = new Baby("Shiloh");
Baby shiloh2 = new Baby("Shiloh");
```

Does `shiloh1 == shiloh2`**?**

References

- The object's location is called a reference
- == compares the references

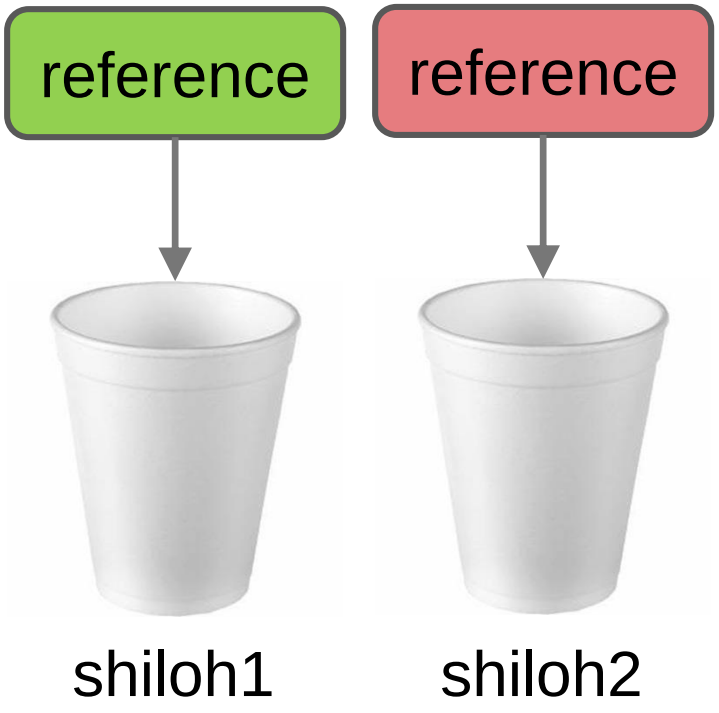
```
Baby shiloh1 = new Baby("Shiloh");
Baby shiloh2 = new Baby("Shiloh");
```

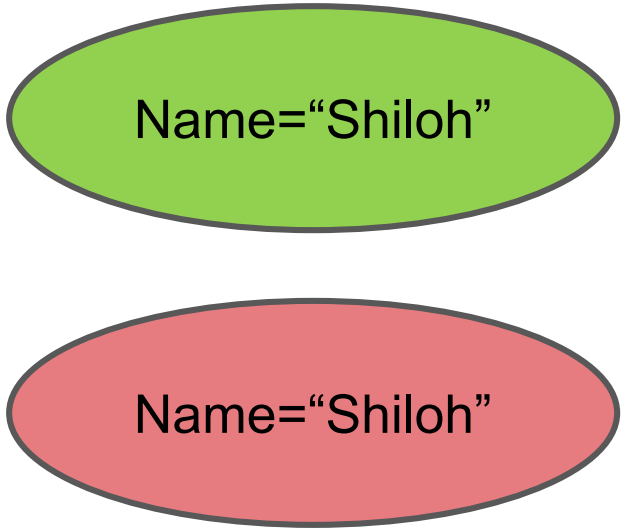
Does shiloh1 == shiloh2?

NO!

References

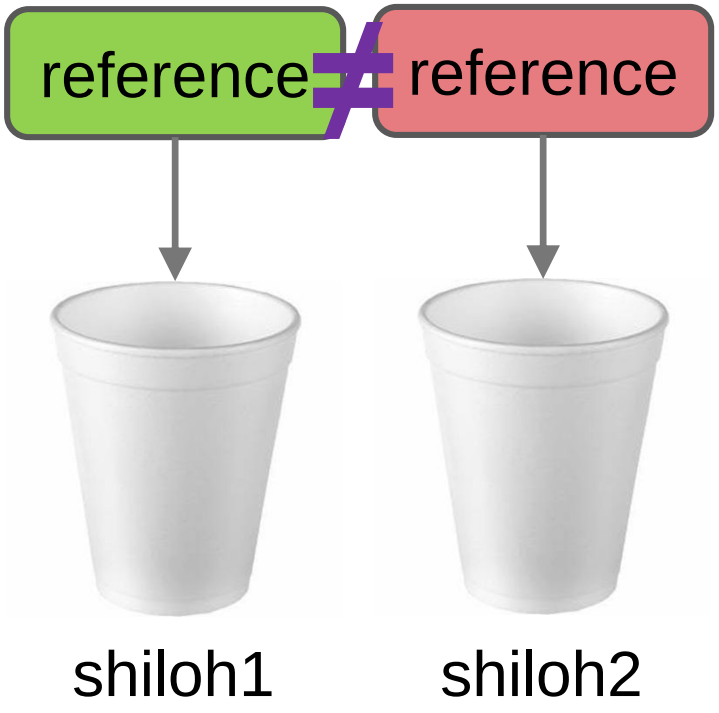
```
Baby shiloh1 = new Baby("Shiloh");
Baby shiloh2 = new Baby("Shiloh");
```

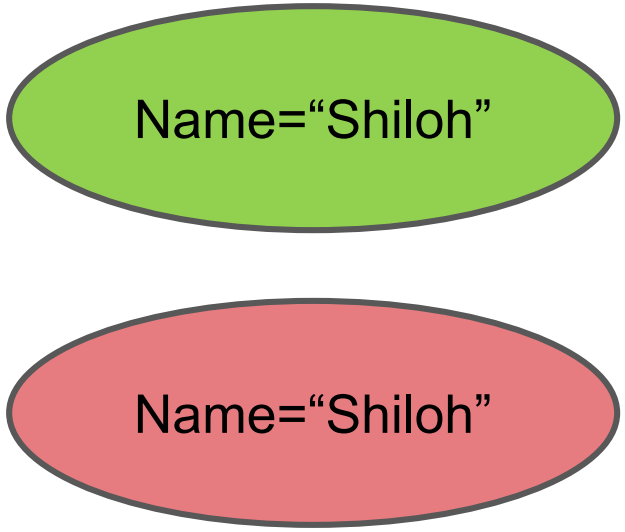




References

```
Baby shiloh1 = new Baby("Shiloh");
Baby shiloh2 = new Baby("Shiloh");
```





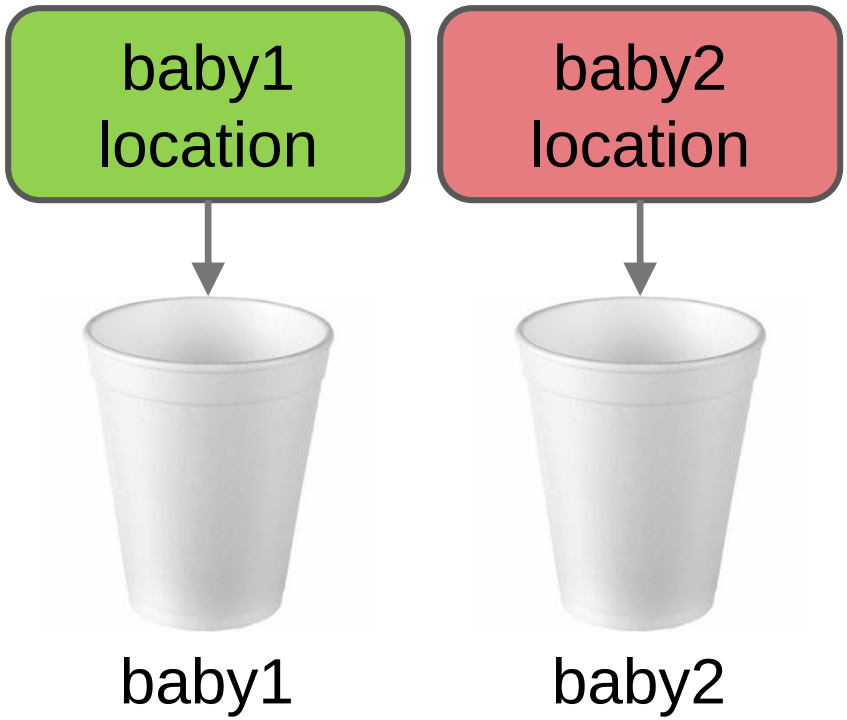
Relations between objects

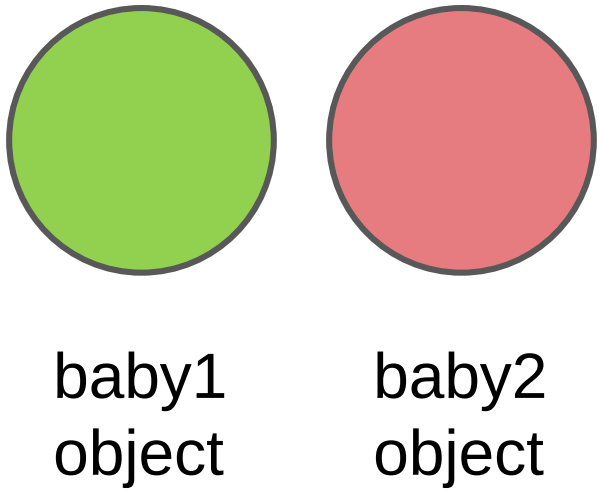
1. **using ==**
   - `shiloh1 == shiloh2`
   - Check two variables reference exactly same
2. **using field**
   - `shiloh1.name == shiloh2.name`
   - Compare a field of the object
3. **using user-defined method**
   - `shiloh1.equals (shiloh2)`
   - Used when the objects are different, but determine the same or not by the fields stored in the object

References

- Using = updates the reference

```
baby1 = baby2;    //?
```

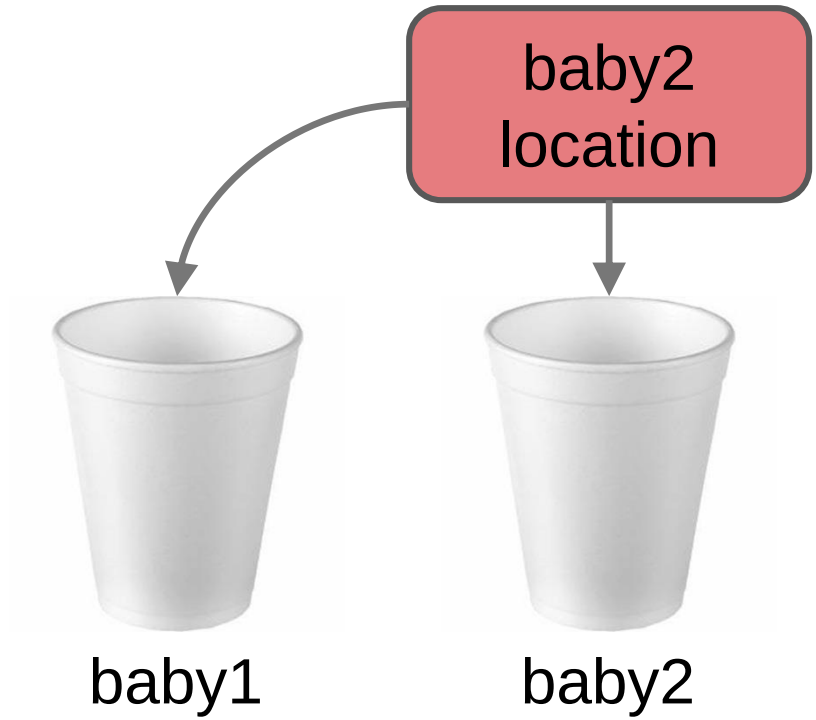


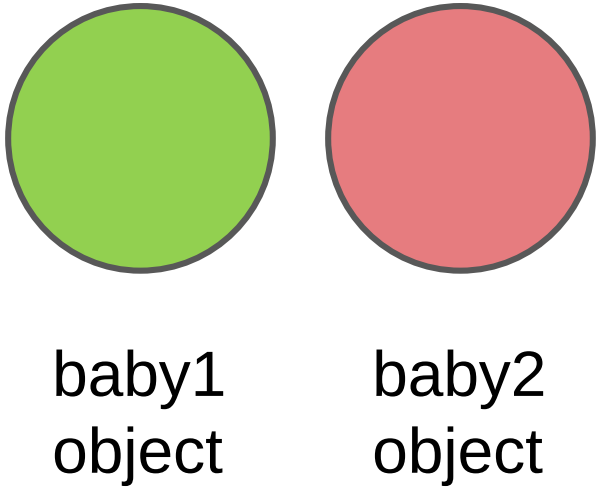


References

- Using = updates the reference

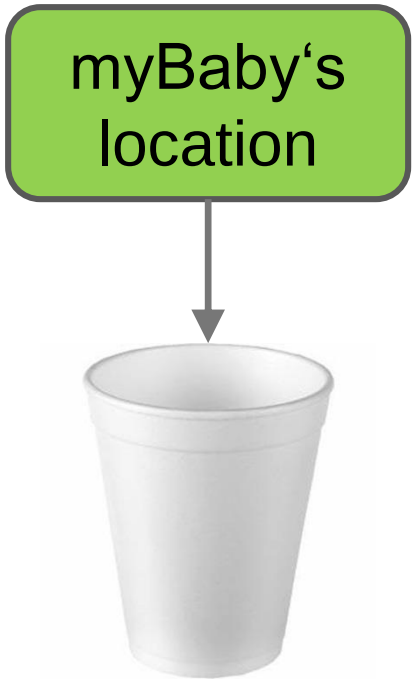
```
baby1 = baby2;
```





References

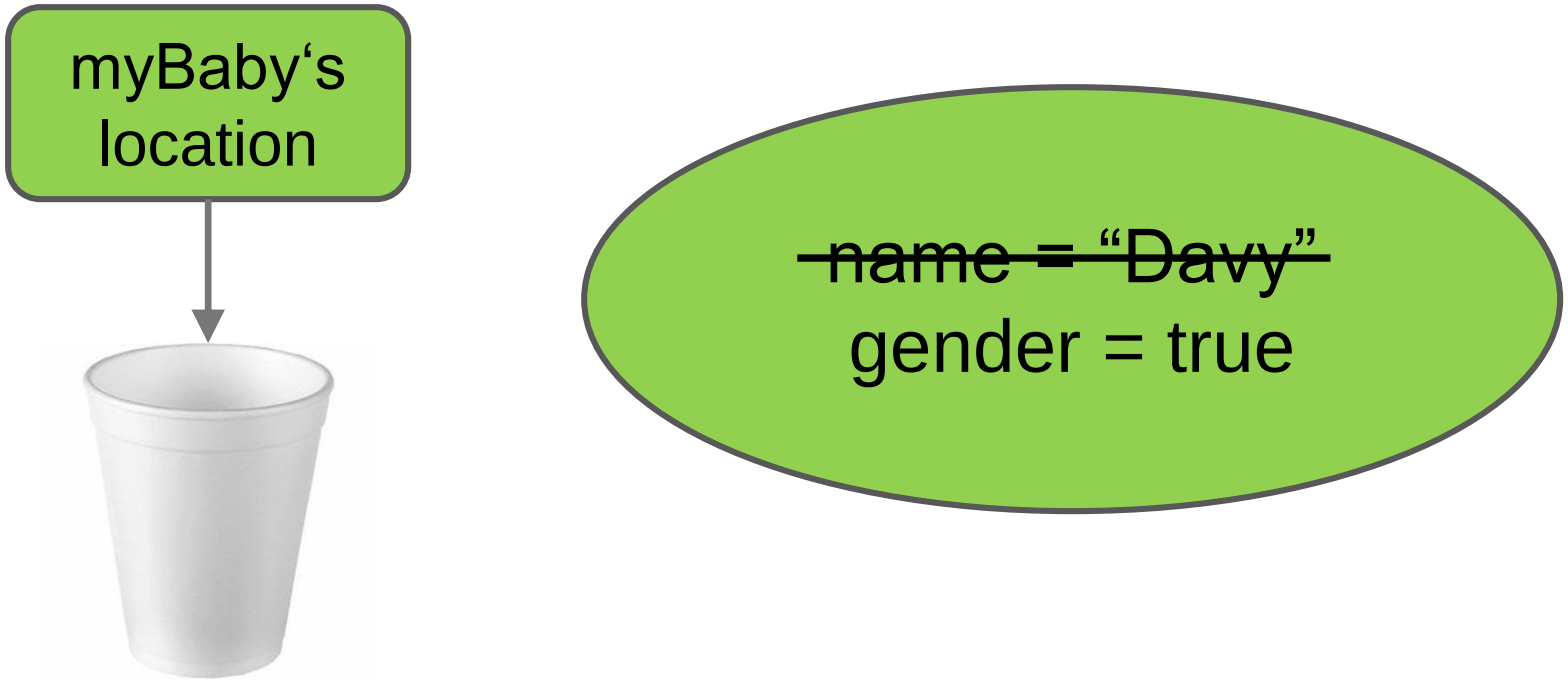
```
Baby myBaby = new Baby("Davy", true);
```



name = "Davy"
gender = true

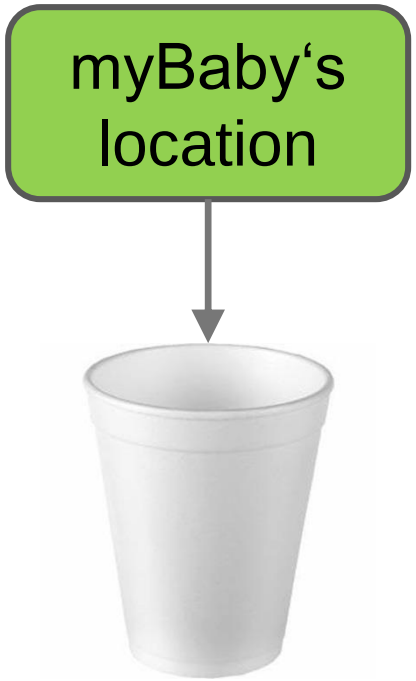
References

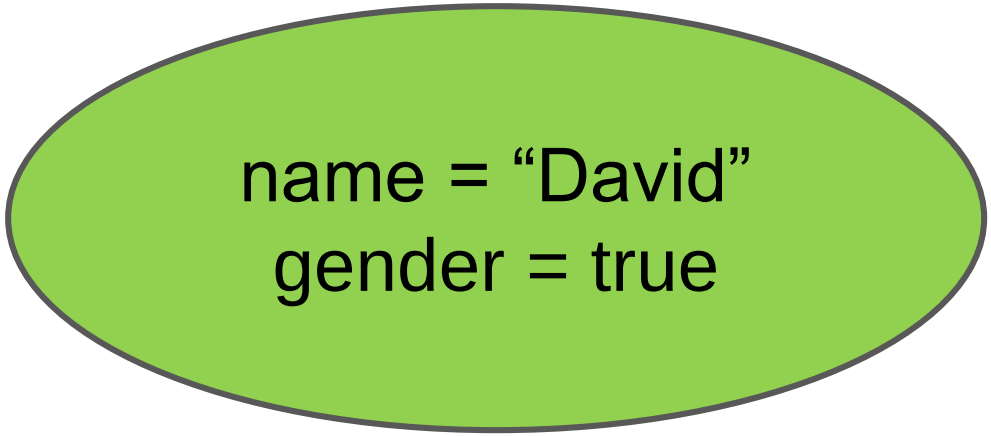
```
Baby myBaby = new Baby("Davy", true);
myBaby.name = "David";
```



References

```
Baby myBaby = new Baby("Davy", true);
myBaby.name = "David";
```





References

- **Using [ ] or .**
  - Follows the reference to the object
  - May modify the object, but never the reference
- **Imagine**
  - Following directions to a house
  - Moving the furniture around
- **Analogous to**
  - Following the reference to an object
  - Changing fields in the object

Self reference

- **Java class has a special way to access itself**

```
class Baby {
  String[] words;
  …
  void remember (String[] words) {
    String word;
    for (word : words) words[top++] = word;          // ???????
  }
}
```

Self reference

- **Java class has a special way to access itself**

```
class Baby {
  String[] words;
  …
  void remember (String[] words) {
    String word;
    for (word : words) this.words[top++] = word;    // !
  }
}
```

Methods and references

```
void doSomething(int x, int[] ys, Baby b) {
  x = 99;
  ys[0] = 99;
  b.name = “99”;
}
…
int i = 0;
int[] j = { 0 };
Baby k = new Baby(“50”, true);
doSomething (i, j, k);
```

i = ? j[0] = ? k.name = ?

ENCAPSULATE

Encapsulation

- In real world, there are huge # of objects and all of them have privacy.

- Electric objects, too!

Public and Private

- **public**
  - Able to access/modify it whoever having the object

```
public class Baby {
  public String nickname;
}

public class Stranger {
  void makeNicknameOf (Baby b) {
    b.nickname = “cuty”;
  }
}
```

Public and Private

- **private**
  - Only the object itself can access it

```
public class Baby {
  private String nickname;
}

public class Stranger {
  void makeNicknameOf (Baby b) {
    b.nickname = “cuty”;          // Error!
  }
}
```

How to access private data?

```
public class Baby {
  private String nickname;

  public void setNickname (String nickname, Object o) {
    // check if the object is instance of [Stranger] or not
    if (o instanceOf Stranger) {
      return;
    }
    this.nickname = nickname;
  }
}

public class Stranger {
  void makeNicknameOf (Baby b) {
    b.setNickname (“cuty”, this);
  }
}
```

Then, what is String in java?

- **Built-in class for handling the sequence of characters in high level abstraction**
- **Usage**

```
String name = “Simon”;
char[] data = {‘s’, ‘i’, ‘n’, ‘g’, ‘l’, ‘e’};
String state = new String (data);

// String concatenate
System.out.println (name + “ is a ”  + state);      // Simon is single

// access the substring with the range of indexes
System.out.println (state.substring (1));           // ingle
System.out.println (state.substring (2,4));         // ngl
```

Then, what is String in java?

- **Built-in class for handling the sequence of characters in high level abstraction**
- **Usage**

```
String name = “Simon”;
char[] data = {‘s’, ‘i’, ‘n’, ‘g’, ‘l’, ‘e’};
String state = new String (data);

// comparison
System.out.println (name == “Simon”);                  // false
System.out.println (name.compareTo (“Simon”));         // true

// get length of a string
System.out.println (name.length ());                   // 5
```

[Lab – Practice #2]

- **Modeling Book and Libraries**
  - class Book {}
  - class Library {}

- **Books can be**
  - borrowed
  - returned

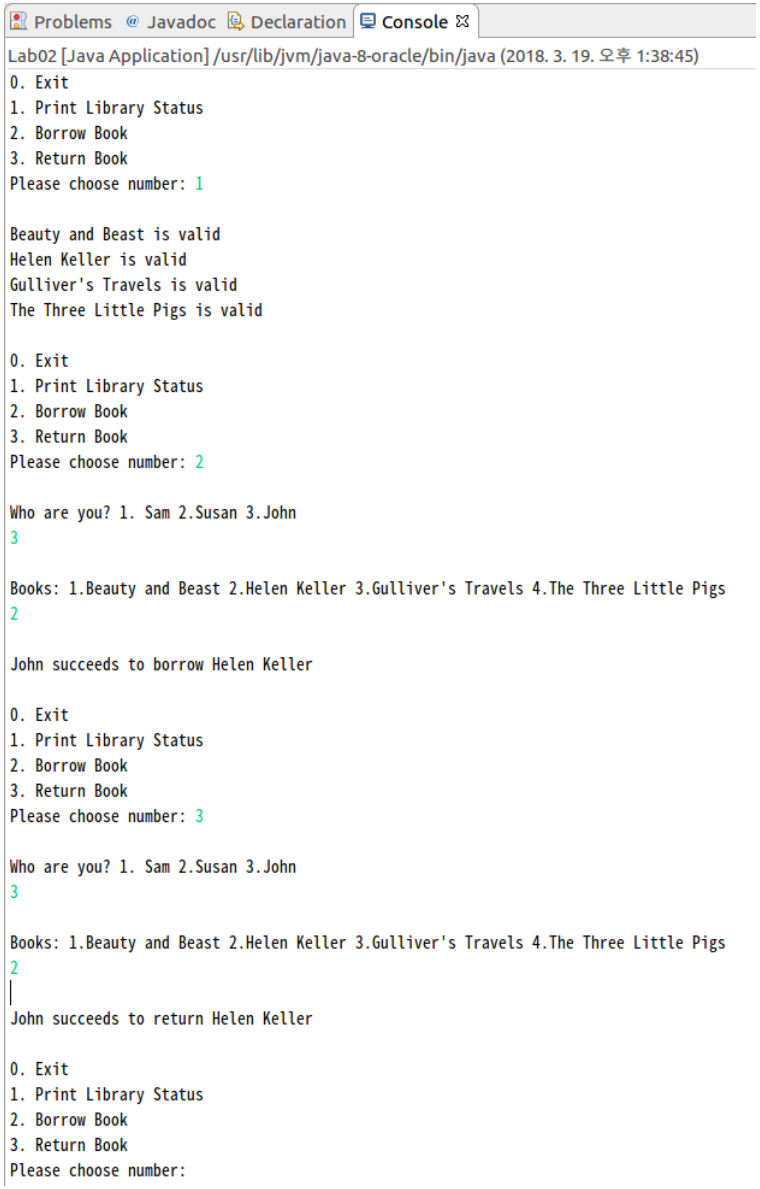
- **Library**
  - keeps ***track*** of books
  - Hint: use Book[]

[Lab – Practice #2]

- **Four books (each book count is one)**
  - Beauty and Beast
  - Helen Keller
  - Gulliver's Travels
  - The Three Little Pigs

- **Three people**
  - Sam
  - Susan
  - John

[Lab – Practice #2]



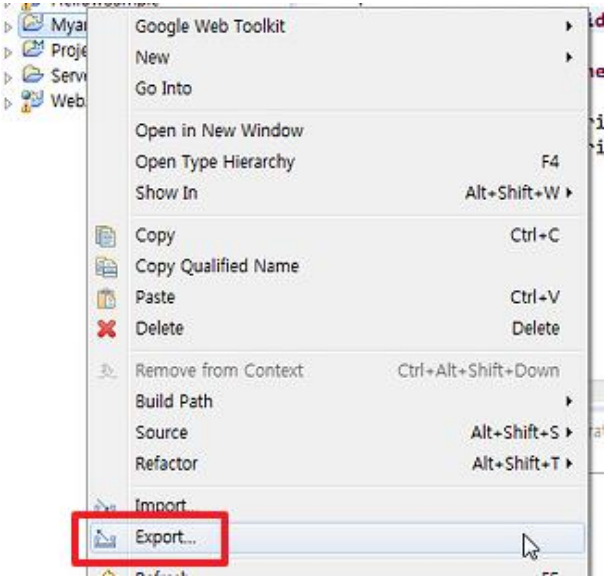
- **Skeleton code is uploaded on i-Campus**
- **Fill in the Book, Library class & main function**
- **Left image is console output example**

[Submit]

- **Upload to i-Campus**
  - Compress your project directory to zip file
  - File name: studentID_lab02.zip

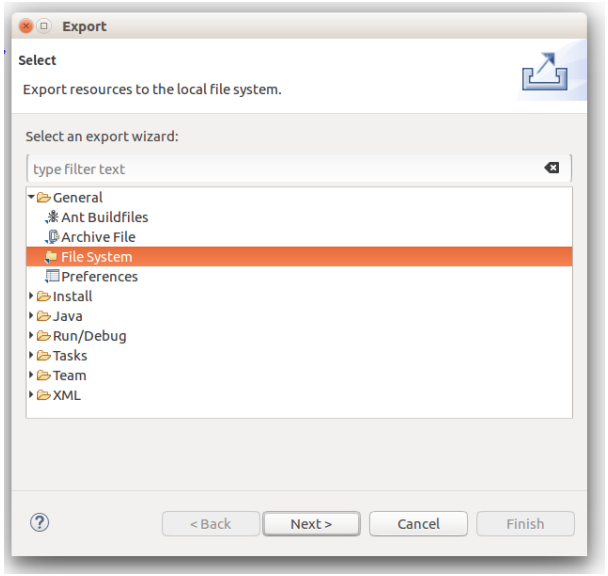
- **Due date**
  - Today 23:59:59
    - Class 42 (3/19 Monday)
    - Class 43 (3/21 Wednesday)
  - Penalty: -10% of each lab score per one day

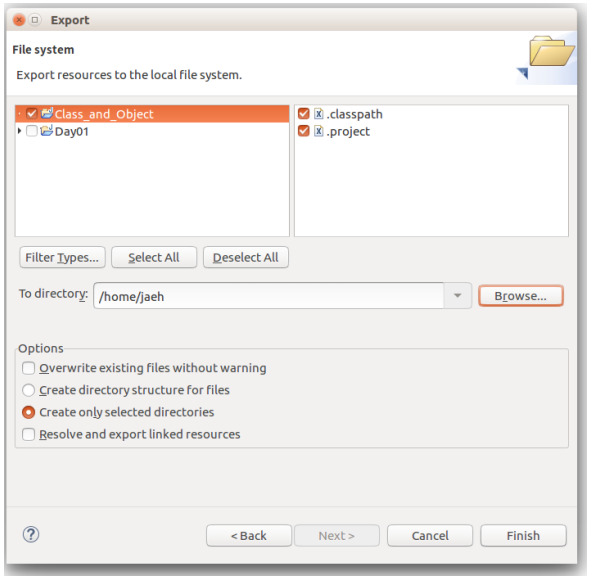
[Project Export]



1. Click mouse right button at project you want to export & choose Export

2. Choose ***General > File system*** and click next button





3. Designate save directory path and click Finish button