Topic 3.1: Objects

© Lauren Himbeault 2024

Learning Goals (Week 3):

- References to Objects
- The clone() method
- Arrays of Objects
- Objects as Method Parameter
- **Understand Java Garbage Collection**
- **Objects in Objects: Safe Creation**
- **Objects in Objects: Safe Method Use**
- **Understand Compartmentalization & Encapsulation Features**

© Lauren Himbeault 2024

Garbage Collection

- Lots of talk of references and ‘new’ memory; runtime stack and heap memory
  - Why don’t we ever run out?

- Java collects ‘garbage’ for us throughout the life of our program
  - But how does it know what to collect?
  - How does Java make sure it isn’t grabbing something we need access to later?

© Lauren Himbeault 2024

Garbage Collection

- Lots of talk of references and ‘new’ memory; runtime stack and heap memory
  - Why don’t we ever run out?

- Java collects ‘garbage’ for us throughout the life of our program
  - Probably should be called recycling since we will recycle the computer space, although the values do get thrown in the ‘garbage’
  - But how does it know what to collect?
  - How does Java make sure it isn’t grabbing something we need access to later?

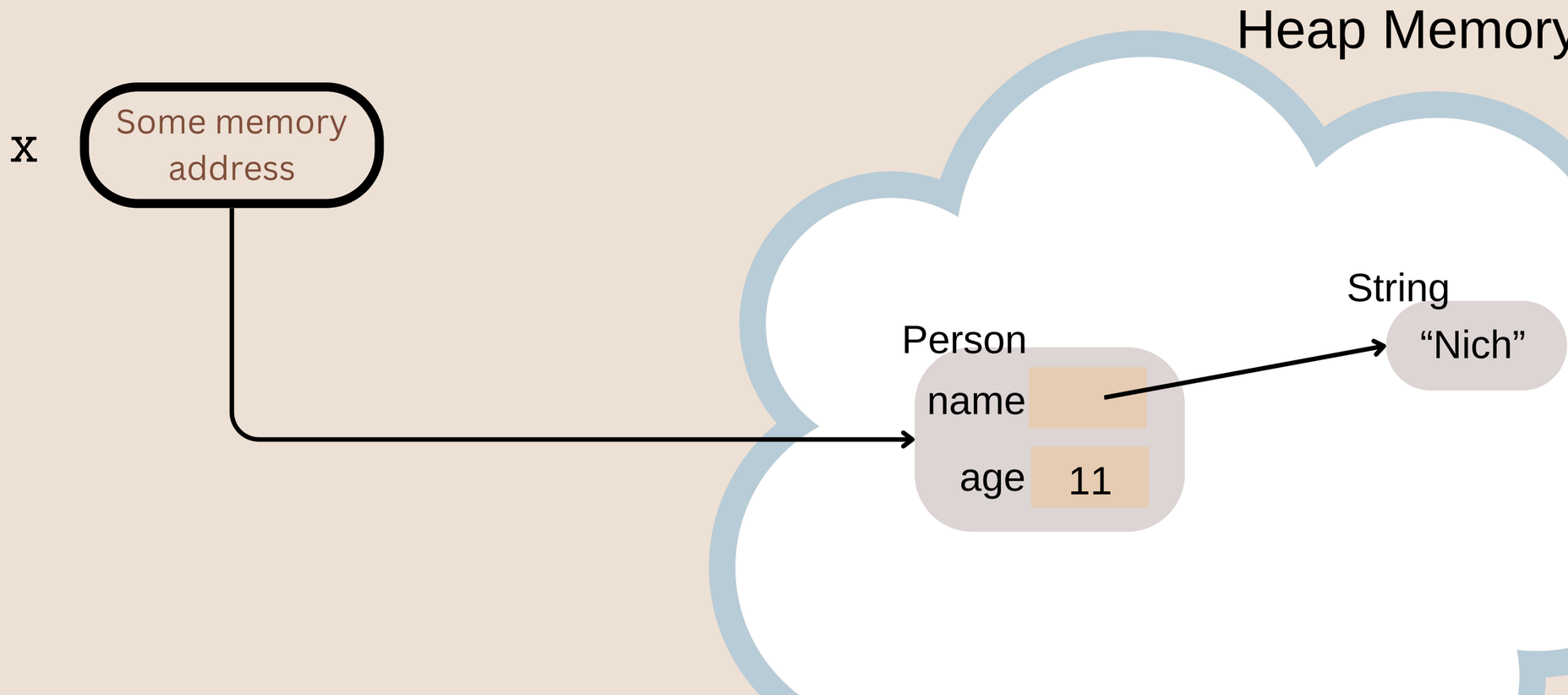
We throw ‘orphans’ in the garbage to make way for new valuable data…

© Lauren Himbeault 2024

Orphans and garbage collection

- When there are no places where the reference to an object is stored, it is no longer usable
  - It's an **'orphan'**

```
Person x = new Person("Nich", 11);
```

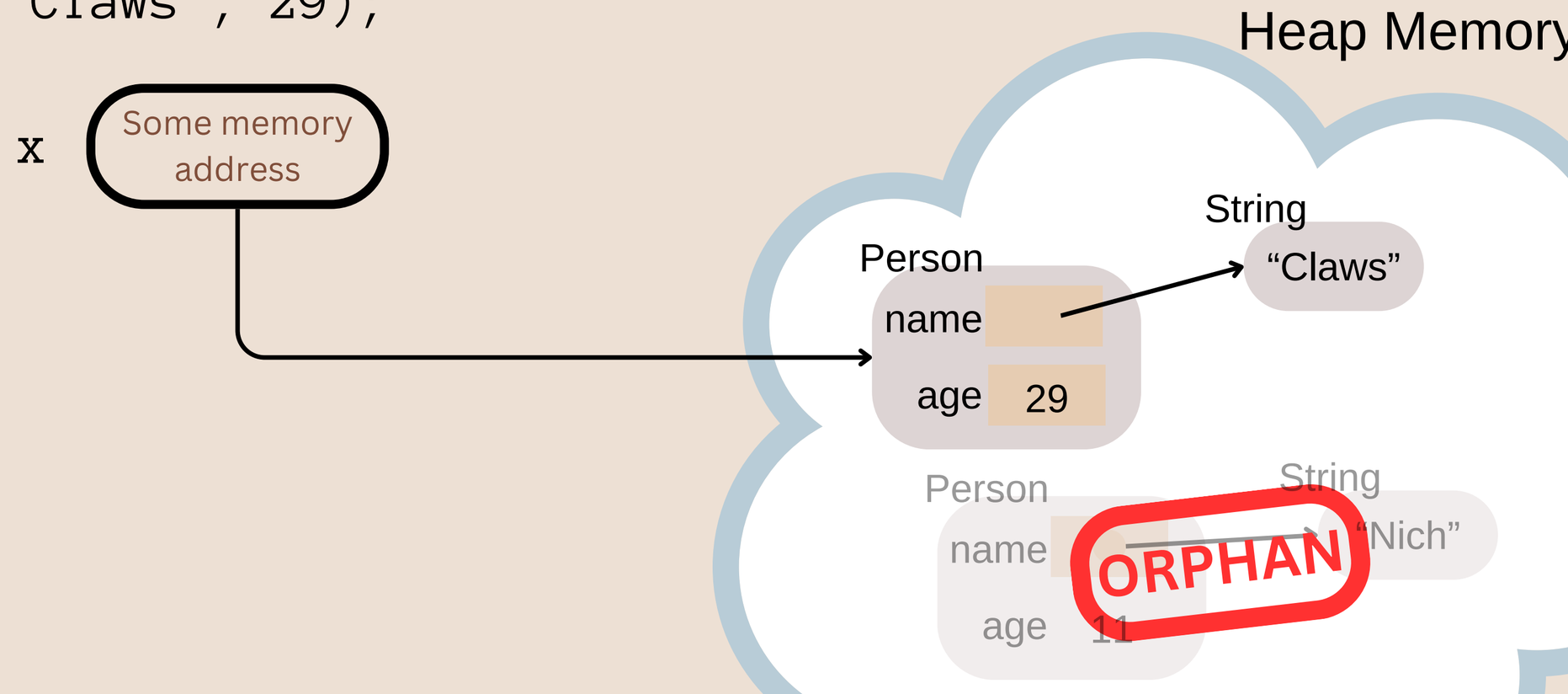


© Lauren Himbeault 2024

Orphans and garbage collection

- When there are no places where the reference to an object is stored, it is no longer usable
  - It's an **'orphan'**

```
Person x = new Person("Nich", 11);
x = new Person("Claws", 29);
```

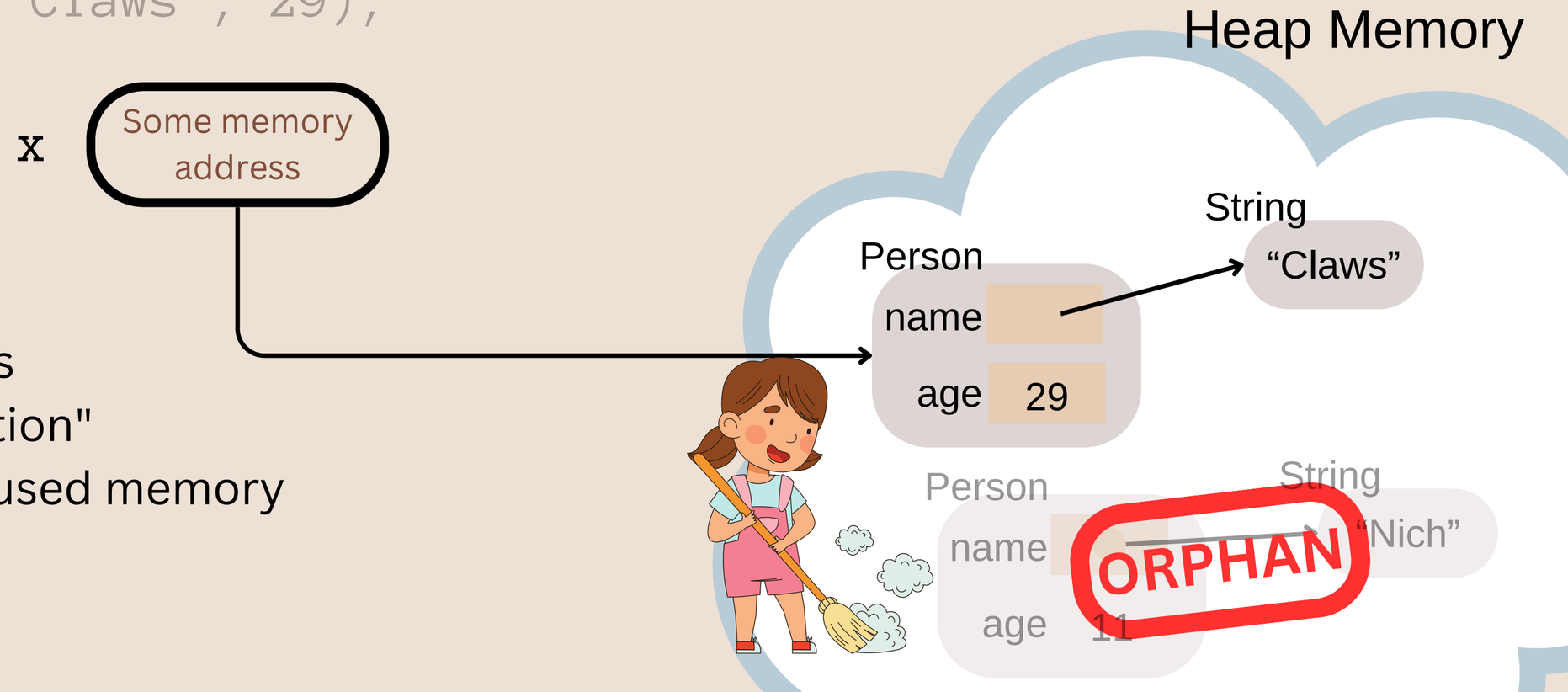


© Lauren Himbeault 2024

Orphans and garbage collection

- When there are no places where the reference to an object is stored, it is no longer usable
  - It's an **'orphan'**

```
Person x = new Person("Nich", 11);
x = new Person("Claws", 29);
```



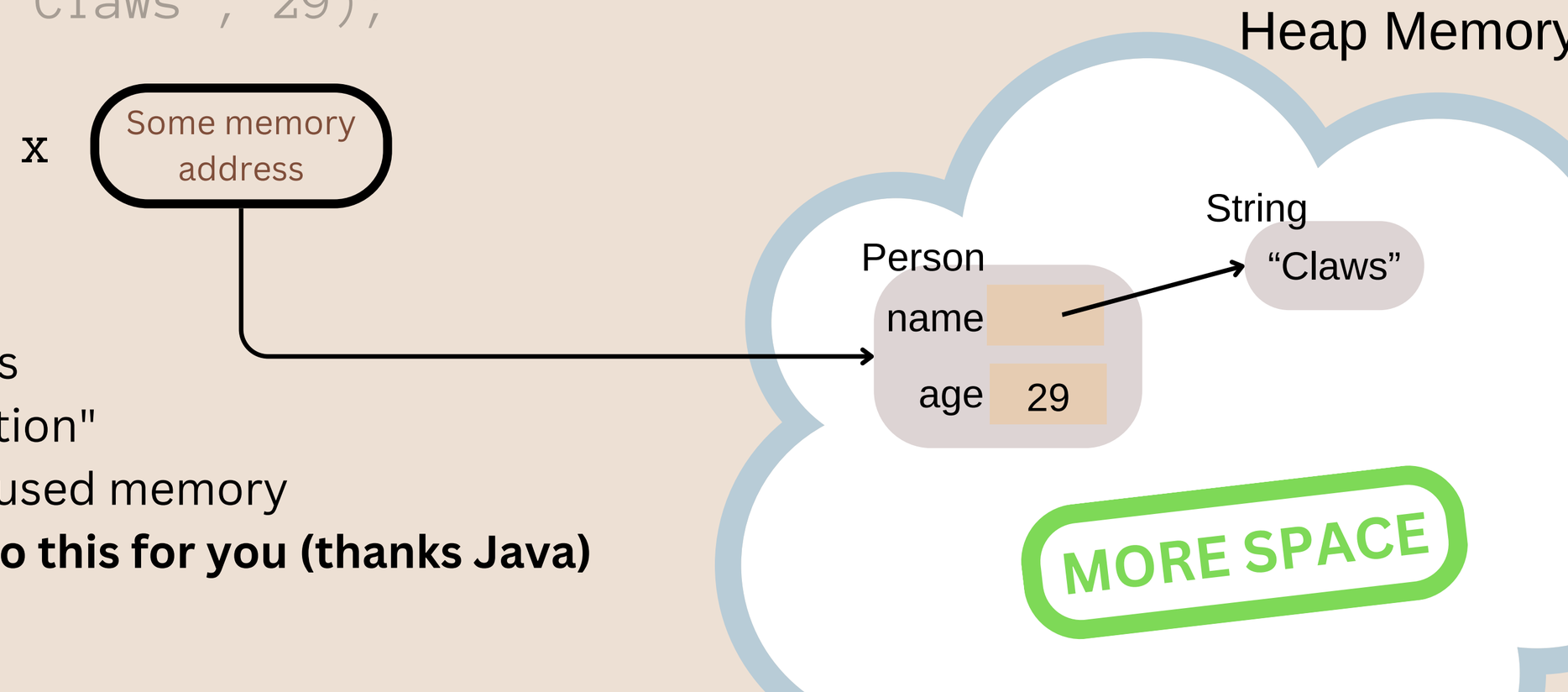
- Java will handle this
  - "garbage collection"
  - frees up any unused memory

© Lauren Himbeault 2024

Orphans and garbage collection

- When there are no places where the reference to an object is stored, it is no longer usable
  - It's an **'orphan'**

```
Person x = new Person("Nich", 11);
x = new Person("Claws", 29);
```



- Java will handle this
  - "garbage collection"
  - frees up any unused memory
- **Not all languages do this for you (thanks Java)**

© Lauren Himbeault 2024

Objects in Objects

- An instance variable in an object can be of any type, including object types (Just like a String in our Person class
- This means they contain a reference to some other object, not the object itself
- **This is extremely common and very powerful**

- Let's change our Person object:

```
// Instance variables
private String name;
private int age;
private Person spouse; // null means no spouse
// how about Person[] children ? Sure. Later.
```

© Lauren Himbeault 2024

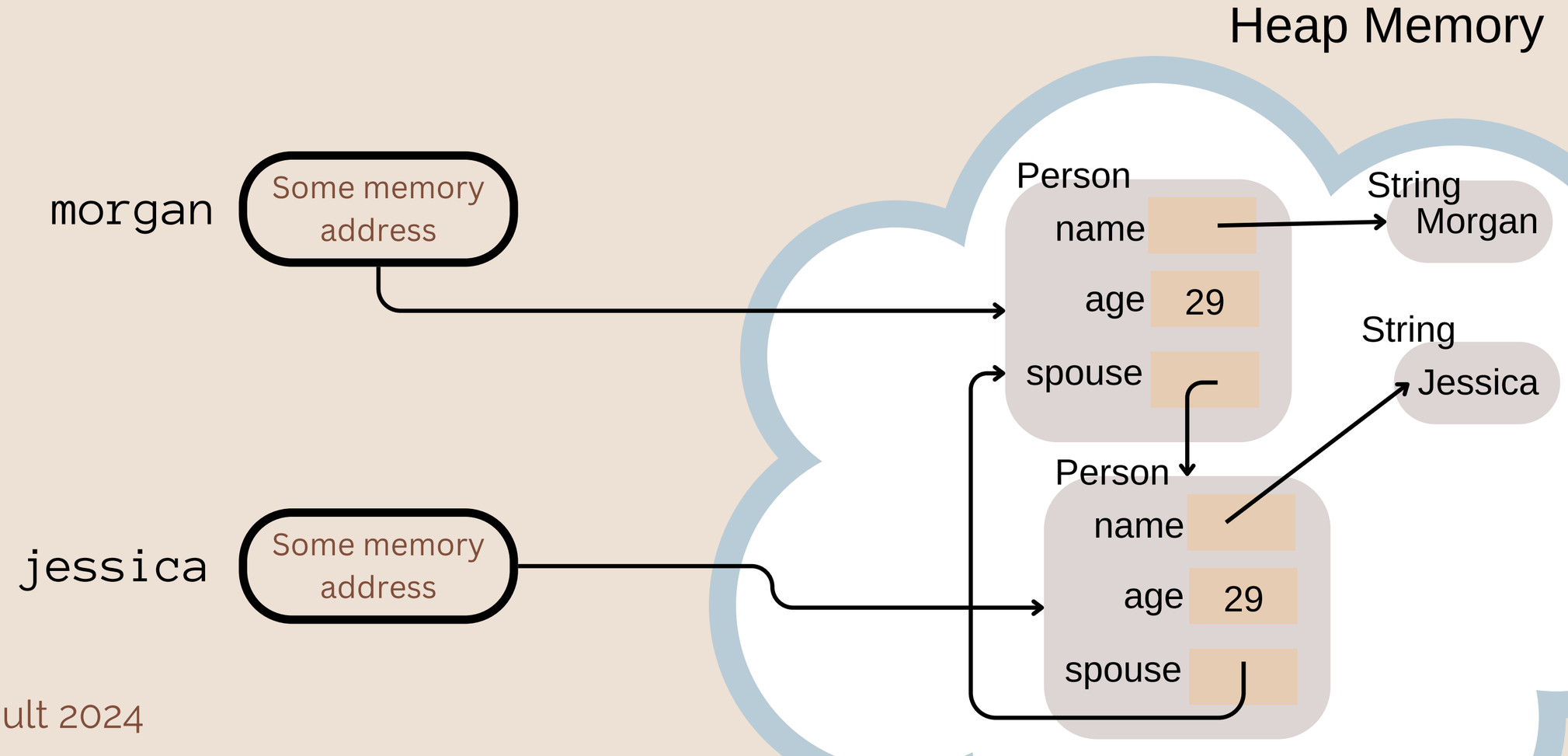
More new stuff for our updated Person

```
// new constructor
public Person(String who, int currentAge, Person otherHalf) {
    name = who;
    age = currentAge;
    spouse = otherHalf;
    //make sure the other person is married, too!
    if(otherHalf != null)
       otherHalf.spouse = this;
} // constructor
```

© Lauren Himbeault 2024

More new stuff for our updated Person

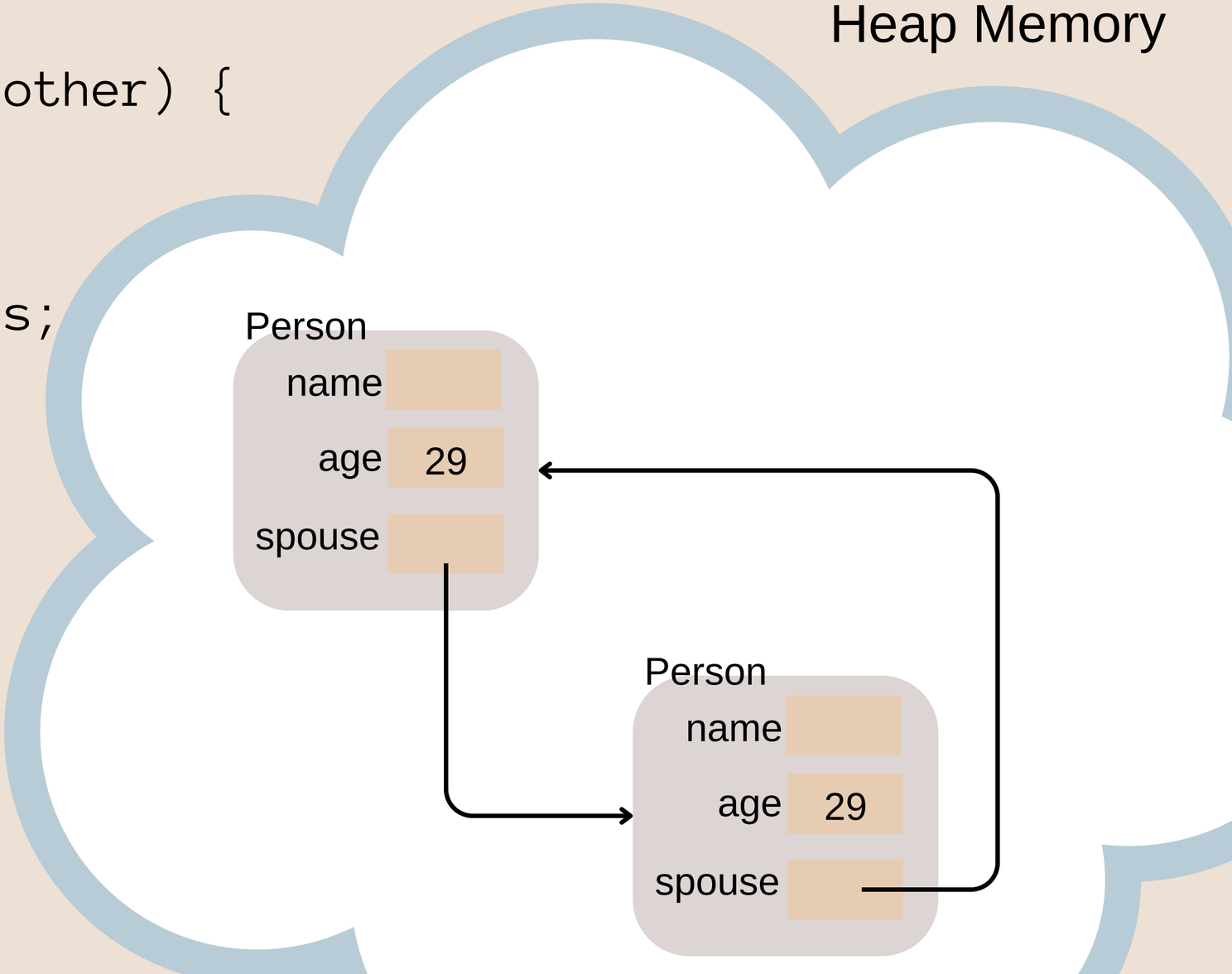
```
Person morgan = new Person(“Morgan”, 28); // spouse starts as null
Person jessica = new Person(“Jessica”, 27, morgan);
```



© Lauren Himbeault 2024

New methods

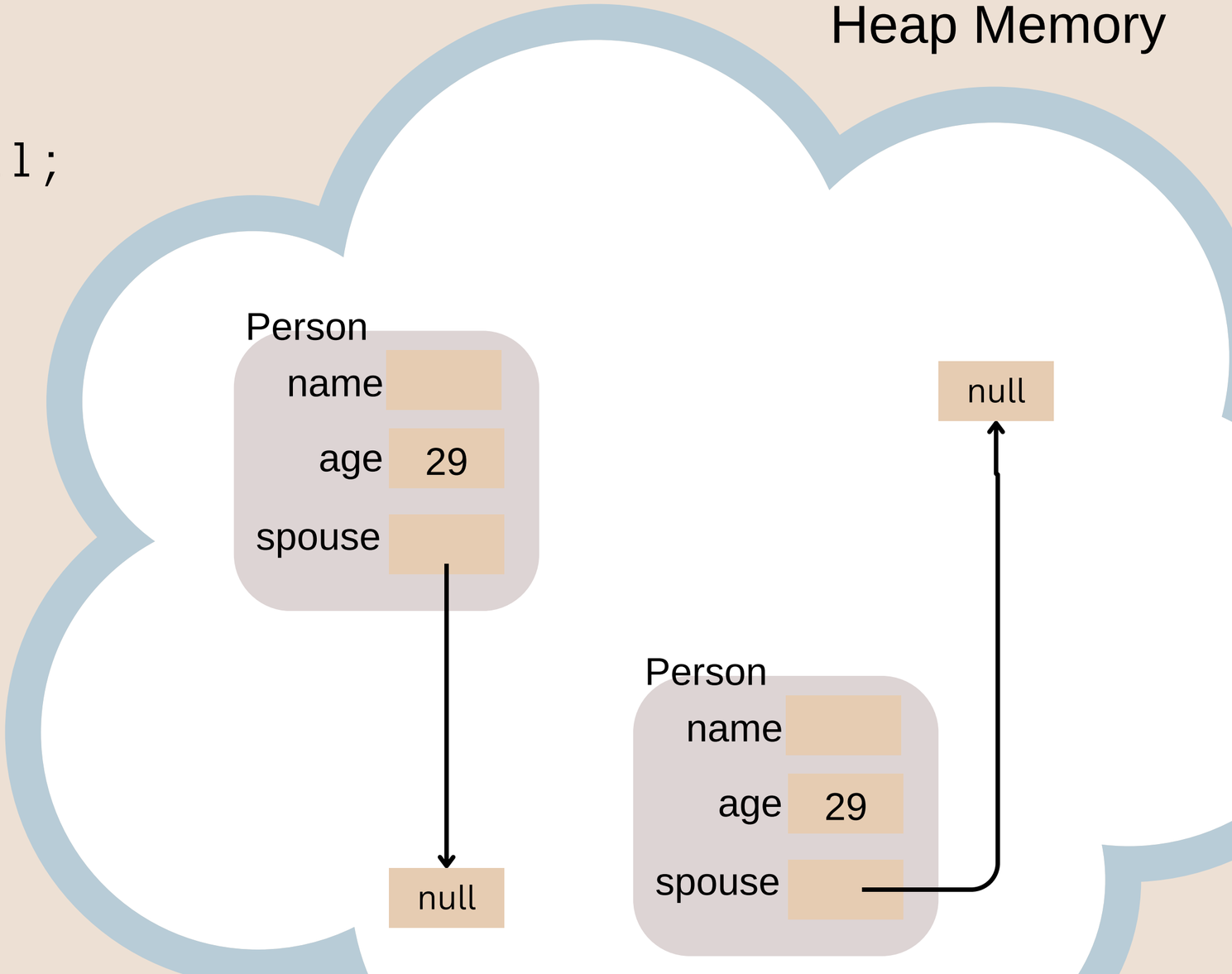
```
public void marries(Person other) {
    spouse = other;
     if (other != null)
         other.spouse = this;
} // marries
```



© Lauren Himbeault 2024

New methods

```
public void divorces() {
    if (spouse != null){
        spouse.spouse = null;
        spouse = null;
    }
} // divorces 😢
```

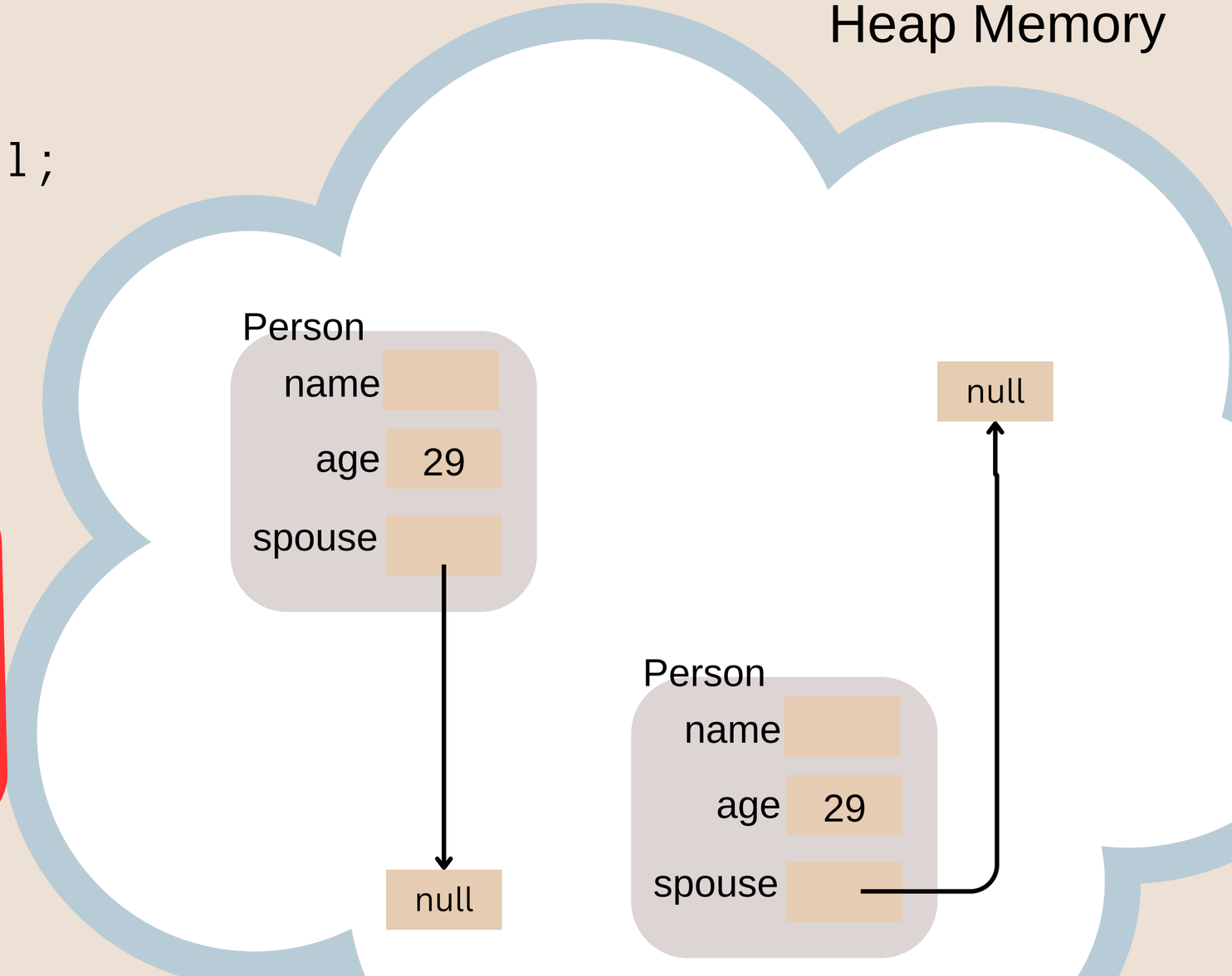


© Lauren Himbeault 2024

New methods

```
public void divorces() {
    if (spouse != null){
        spouse.spouse = null;
        spouse = null;
    }
} // divorces 😢
```

Order of operations is important here!
If you did it the other way around:
spouse = null;
spouse.spouse = null;
You would get a null pointer exception!



© Lauren Himbeault 2024

Some more Helpers

```
public boolean isMarried() {
   return spouse != null; //don’t use an IF here, useless!
}

public Person getSpouse() {
   return spouse;
}
```

- We might want to update the toString method to print the name of the spouse…
  - How would we do that?

© Lauren Himbeault 2024

Passing an object to a method

- We’ve seen earlier that it works the same way as if it was a primitive type
  - you just declare the type of the parameter (Person for example)

```
public void marries(Person other) {
    spouse = other;
    if (other != null)
        other.spouse = this;
} // marries
```

© Lauren Himbeault 2024

Passing an object to a method

- But how does passing a parameter really work in Java?
  - Java always passes a **copy** of the **value** to a method, not the variable itself

- When passing a **primitive** type, a **copy of the value** is passed to the method
- When passing an **object**, we are storing an **address**(reference) so a **copy of the reference** is passed to the method

© Lauren Himbeault 2024

Passing Primitives

```
public static void main (String[] args) {
   int x = 5;
   changeValue(x);
   System.out.println(x); // What is printed?
}

public static void changeValue(int x) {
   x += 10; // x here is just a copy of the value that was passed to the method!
}
```

Passing Objects

© Lauren Himbeault 2024

Passing Primitives

```
public static void main (String[] args) {
   int x = 5;
   changeValue(x);
   System.out.println(x); // What is printed?
}

public static void changeValue(int x) {
   x += 10; // x here is just a copy of the value that was passed to the method!
}
```

Passing Objects

```
public static void main (String[] args) {
   Person p = new Person(“George”, 65);
   changeValue(p);
   System.out.println(p);// What is printed?
}
public static void changeValue(Person p) {
   p = new Person(“Janet”, 48);
   // p here is just a copy of the reference that was passed to the method!
   // Modifying where it points to does not affect the initial paassed reference
}
```

© Lauren Himbeault 2024

Passing Objects

```
// In a class:
public static void main (String[] args) {
   Person p = new Person(“Sayam”, 65);
   changeValue(p);
   System.out.println(p); // What is printed?
}

public static void changeValue(Person p) {
    p.haveBirthday();
   // not reassigning p but rather altering the information stored at the reference p
  // this now acts like an alias, altering the origin main() object
}
```

© Lauren Himbeault 2024

One final step

- Let’s add a list of children to our Person object
- **But a list of people is a different thing from a Person…**
  - It has its own unique actions
    - Print the whole list
    - Search for a certain Person in the list
    - Add/remove from the list
    - (remove!? This example is becoming very dark…)

© Lauren Himbeault 2024

Best way to handle this?

- There should be a separate **PersonList** class, which will handle all these operations

- Write a **PersonList** class with:
  - A “partially-filled array” of Person
  - Use a generous fixed size
  - A constructor to make an empty list
  - Methods addPerson, removePerson, and toString

© Lauren Himbeault 2024

Connecting these classes

- Add an instance variable **PersonList** children to the Person class
  - Adjust the constructors as needed
  - Provide methods in **Person**, **that will make use of the methods in PersonList**
    - void addChild(Person)
    - String getListOfChildrenString( )
- **Let’s build this!**

© Lauren Himbeault 2024

Why not just use an Array? Why do we even need PersonList?

- **Separation of Concerns (SoC)/Compartmentalization, Resuability**, & Encapsulation!
- **SoC**
  - all those behaviours we listed above that we can do with a List of people **do not** belong in **main**() and **do not** belong in **Person** (singular)
  - This needs its own space to live
  - Dividing the work between the different objects: PersonList will take care of all operations that can be done on its data (the partially-filled array) that Person doesn't care about
- **Reusability**
  - PersonList is a general class that can be reused every time you need a list of Person objects
  - Can be used for other purposes than list of children:
    - List of employees
    - List of students
    - Etc.

© Lauren Himbeault 2024

Why not just use an Array? Why do we even need PersonList?

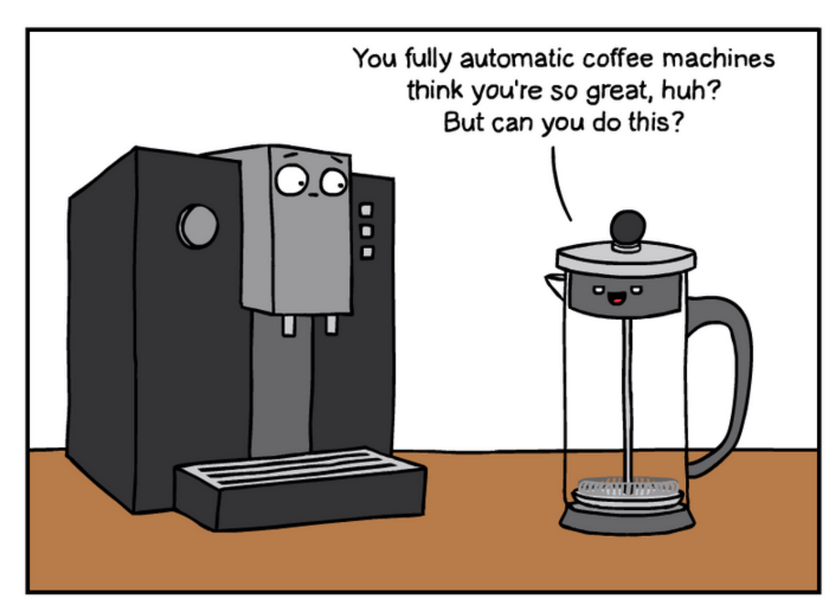
- Separation of Concerns (SoC)/Compartmentalization, Resuability, **& Encapsulation**!
- **Encapsulation**
  - The original Person object won't have to worry about how PersonList manages the list, and just use the public methods offered by PersonList (encapsulation)
  - Hide away information/data where appropriate

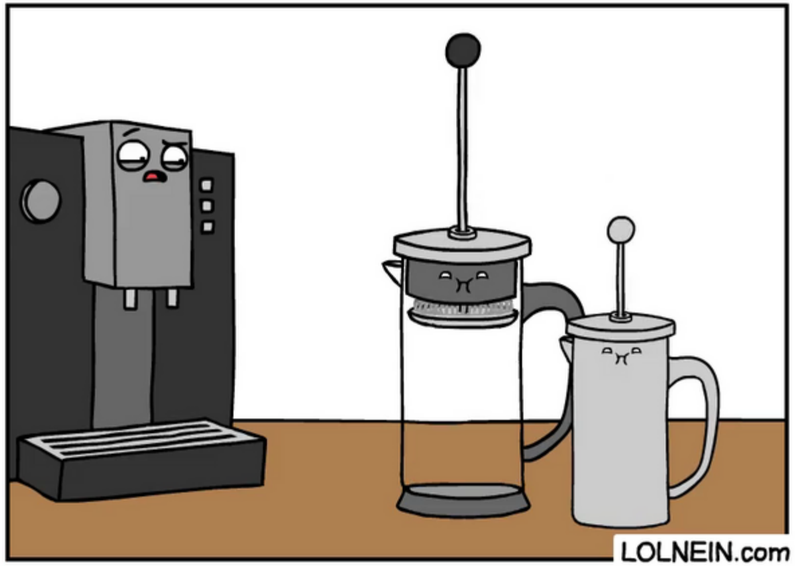
© Lauren Himbeault 2024

Why not just use an Array? Why do we even need PersonList?

- Separation of Concerns (SoC)/Compartmentalization, Resuability, **& Encapsulation**!
- **Encapsulation**
  - The original Person object won't have to worry about how PersonList manages the list, and just use the public methods offered by PersonList (encapsulation)
  - Hide away information/data where appropriate
- Remember **Encapsulation** means hiding away things that don't concern you.

You don't need to know how the coffee machine works to press the button and get coffee.





LOLNein

© Lauren Himbeault 2024

Pause & Practice

Recipe Management System
Create a simple recipe management system. This system will involve three classes: `**Recipe**`, `**Ingredient**`, and `**IngredientList**`.

Ingredient Class: This class represents an ingredient used in a recipe. Each ingredient should have at least two variables: `**name**` (String) and `**quantity**` (String, e.g., "2 cups" or "100g"). Include appropriate **constructors**, **getters**, and **setters**.

IngredientList Class: This class manages a list of `**Ingredient**` objects. It should allow **adding** and **removing** ingredients, and retrieving the list of ingredients. Implement methods like `**addIngredient**(Ingredient ingredient)`, `**removeIngredient**(String ingredientName)`, and `**getIngredients**()`.

Recipe Class: This class represents a **recipe**. Each recipe should have a **name** (String) and an `**IngredientList**`. Implement methods to **add** and **remove** ingredients from the recipe, get the **recipe name**, and a toString that prints the name and the full ingredient list (including the list of ingredients with their quantities) **nicely**.

Create a main method to demonstrate the use of these classes. Instantiate a `Recipe` object, add a few `Ingredient` objects to it, and display the complete recipe.

© Lauren Himbeault 2024

Reference:
Assistance in formulating the instructions for a Java practice question provided by ChatGPT, an AI developed by OpenAI, specialized in natural language understanding and generation.