Advanced Java Programming

Part 2: references and objects, shallow vs. deep copies, automatic garbage collection, parameter passing

James Tam

Review: Previous Class

- What you have learned in your prerequisite class: some variables directly contain data:

```
num1 = 12
num2 = 3.5
ch = 'a'
```

- What you may have learned your prerequisite class: some variables 'refer' to other variables.

```
list = []
list = [1,2,3]
```

James Tam

Review: This Class

- In Java when you use objects and arrays there are two things involved:
  - Reference
  - Object (or array)
- Example with an object

```
Person charlie;   // Creates reference to object
charlie = new Person("Sheen");  // Creates object
```

- Example with an array

```
double [] salaries; // Creates reference to array
salaries = new double[100];  // Creates array
```

- Normally a newly created reference contains a 'null' value (meaning it refers to 'nothing').
- Roughly equivalent to:

```
-charlie = null;
```

James Tam

Review: This Class

- Why is it important to know that a reference and what the reference refers two are separate?
- **Name of the folder containing the online example**: `4referencesVsObjects`

James Tam

Addresses And References

- Real life metaphor: to determine the location that you need to reach the 'address' must be stored (electronic, paper, human memory)

- Think of the delivery address as something that is a 'reference' to the location that you wish to reach.
  - Lose the reference (electronic, paper, memory) and you can't 'access' (go to) the desired location.

James Tam

Addresses And References

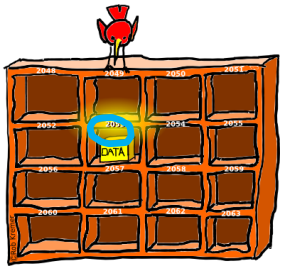
- A reference to an array does not directly contain the contents of the array
  - Instead the reference contains the address ("refers to") of the array

James Tam

Recap: Variables

- Variables are a 'slot' in memory that contains 'one piece' of information.

```
num = 123
```



- Normally a location is accessed via the name of the variable.
  - Note however that each location is also numbered!
  - This is the address of a memory location.

Image: Curtesy of Rob Kremer

References And Objects

- **Name of the folder containing the complete example** : `5referenceExamples`

```
public class Person
{
    private String name;
    public Person() {  name = "none";  }

    public Person(String newName) {  setName(newName);
    }

    public String getName() {  return(name);  }

    public void setName(String newName) {
        name = newName;
    }
}
```

References And Objects (2)

- In `main()`:

```
Person bart;
Person lisa;

bart = new Person("bart");
System.out.println("Bart object name: " + bart.getName());

lisa = bart;
bart = new Person("lisa");
System.out.println("Bart object name: " + bart.getName());
System.out.println("Lisa object name: " + lisa.getName());
```

```
Bart object name: bart
```

```
Bart object name: lisa
```

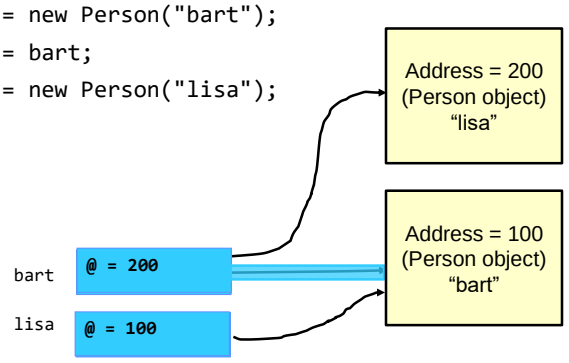
```
Lisa object name: bart
```

James Tam

References And Objects (3)

- What happened?

```
Person bart;
Person lisa;
bart = new Person("bart");
lisa = bart;
bart = new Person("lisa");
```



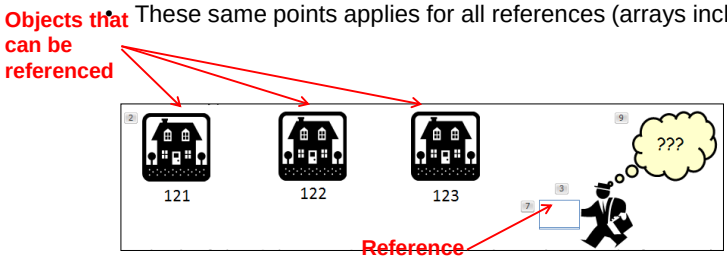
James Tam

References And Objects (4)

```
Person bart;
Person lisa;
bart = new Person("bart");
lisa = bart;
bart = new Person("lisa");
```

Note:

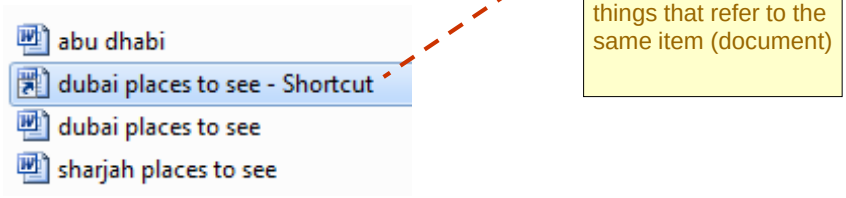
- The object and the reference to the object are separate e.g., 'bart' originally referenced the 'bart object' later it referenced the 'lisa object'
- The only way to access the object is through the reference.
- These same points applies for all references (arrays included)



James Tam

Shallow Copy Vs. Deep Copies

- Shallow copy (new term, concept should be review)



- **New term**:
  - Copy the address from one reference into another reference
  - Both references point to the same location in memory

James Tam

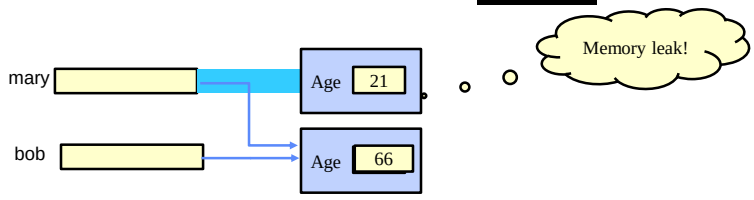
Shallow Copy Vs. Deep Copies (2)

- **Name of the folder containing the complete example**:
  `6shallowVsDeep`

```
Person mary = new Person(21);
Person bob = new Person(12);
System.out.println(mary.age + " " +
   bob.age);
mary = bob;   // Shallow;
bob.age = 66;
System.out.println(mary.age + " " +
   bob.age);
```

```
21 12
```

```
66 66
```



James Tam

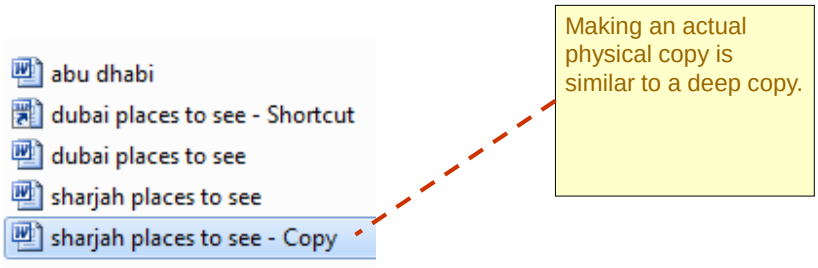
New Term: Memory Leak

- When memory that was used by a program is no longer needed but is not freed up for other programs.

James Tam

Shallow Copy Vs. Deep Copies (3)

- Deep copy (new term, concept should be review)



  - **New term, deep copy**:
    - It's not the addresses stored in the references that's copied
    - Instead the data referred to by the references are copied
  - After the copy each reference still refers to a different address (the address refers to a data variable)

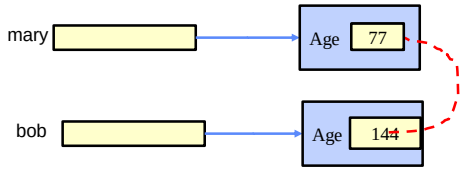
James Tam

Shallow Copy Vs. Deep Copies (4)

- **Name of the folder containing the complete example** : `6shallowVsDeep`

```
// Mary still 66
bob = new Person(77);
mary.age = bob.age;     // Deep
bob.age = 144;
System.out.println(mary.age + " " +
   bob.age);
```

```
77 144
```

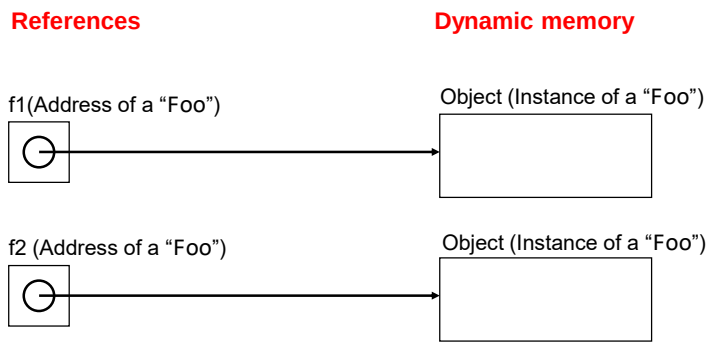


James Tam

Automatic Garbage Collection Of Java References

- Dynamically allocated memory is automatically freed up when it is no longer referenced (Foo = a class) e.g.,

```
Foo f1 = new Foo();
Foo f2 = new Foo();
```

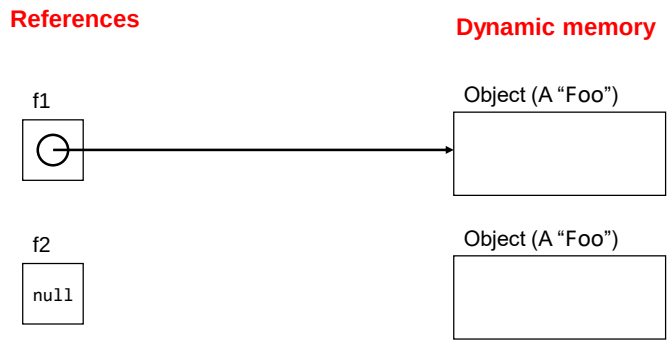


James Tam

Automatic Garbage Collection Of Java References (2)

- Dynamically allocated memory is automatically freed up when it is no longer referenced e.g.,

```
f2 = null;
```



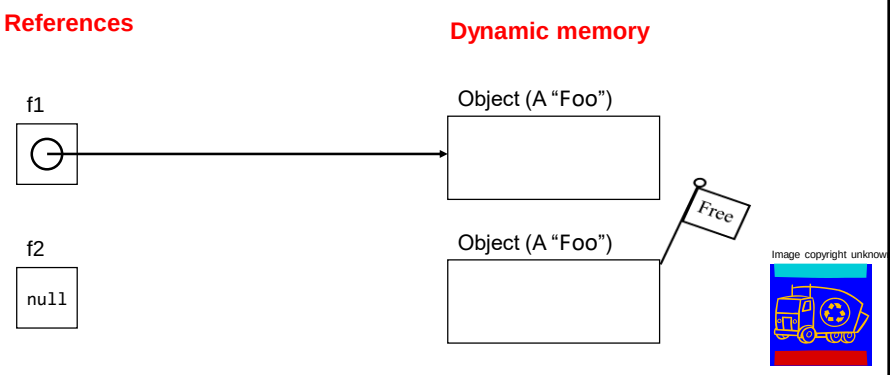
James Tam

Automatic Garbage Collection Of Java References (3)

- Dynamically allocated memory is automatically freed up when it is no longer referenced e.g.,

  `f2 = null;`

  - Recall that a `null` reference means that the reference refers to nothing, it doesn't contain an address).



James Tam

Caution: Not All Languages Provide Automatic Garbage Collection!

- Some languages do not provide automatic garbage collection (e.g., C, C++, Pascal).
- In this case dynamically allocated memory must be manually freed up by the programmer.
- **New term**: Memory leak: memory that has been dynamically allocated (such as via the Java 'new' keyword') but has not been freed up after it's no longer needed.
  - Memory leaks are a sign of poor programming style and can result in significant slowdowns.

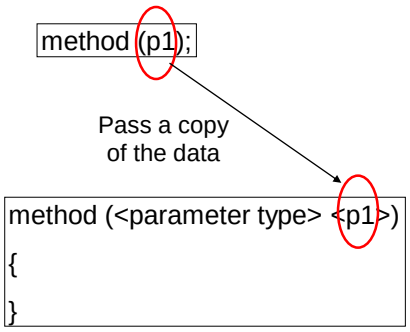
James Tam

Methods Of Parameter Passing

- **New term**: Pass by value
  - The data stored (the "*value*" stored) in the parameter is copied
- **New term**: Pass by reference
  - Pass the address of the parameter
  - This allows references to the parameter inside the method (the method has a "*reference*" to the original parameter).

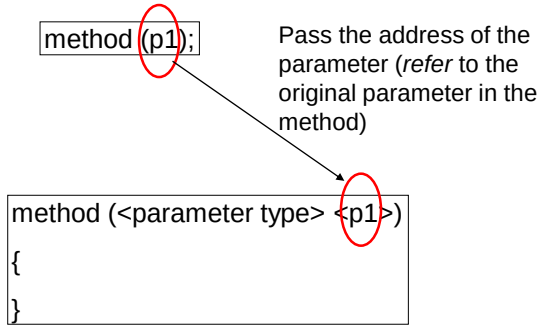
James Tam

Passing Parameters As Value Parameters



James Tam

Passing Parameters As Reference Parameters



James Tam

Which Parameter Passing Mechanism Is Used?

Passed by value

- All 'simple' built in types:
  - Integers (`byte`, `short`, `int`, `long`)
  - Floating point (`float`, `double`)
  - Character (`char`)
  - Boolean (`boolean`)

Pass by reference

- Objects
- Arrays
- (That is, anything that consists of a reference and the item referenced).

James Tam

Parameter Passing Example

- **Name of the folder containing the complete example** : `7parameters`

Class Person

```
public class Person {
    private int age;
    private String name;

    public Person() {
        age = -1;
        name = "none";
    }

    public int getAge() {
        return(age);
    }

    public String getName() {
        return(name);
    }
```

Class Person (2)

```
    public void setAge(int anAge) {
        age = anAge;
    }

    public void setName(String aName) {
        name = aName;
    }
}
```

James Tam

Class ParameterExample

```
public class ParameterExample
{
    public void modify(Person aPerson, int aNum)
    {
        aPerson.setName("Eric Cartman");
        aPerson.setAge(10);
        aNum = 888;
        System.out.println("Person inside modify()");
        System.out.println(aPerson.getName() + " " +
                           aPerson.getAge());
        System.out.println("Number inside modify()");
        System.out.println(aNum);

    }
}
```

Modifies parameters here

James Tam

The Driver Class

```
public class Driver
{
    public static void main(String [] args)
    {
        int num = 13;
        Person aPerson = new Person();
        ParameterExample pe = new ParameterExample();

        System.out.println("Person in main() before edit");
        System.out.println(aPerson.getName() + " " +
                           aPerson.getAge());
        System.out.println("Number inside main() before edit");
        System.out.println(num);
        System.out.println("---
```

```
Person in main() before edit
none -1
Number inside main() before edit
13
```

The Driver Class (2)

```
        pe.modify(aPerson,num);
        System.out.println("----------");
```

```
public void modify(Person aPerson, int aNum)
{
  aPerson.setName("Eric Cartman");
  aPerson.setAge(10);
  aNum = 888;
```

```
Person inside modify()
Eric Cartman 10
Number inside modify()
888
```

```
        System.out.println("Person in main() after edit");
        System.out.println(aPerson.getName() + " " +
                           aPerson.getAge());
        System.out.println("Number inside main() after edit");
        System.out.println(num);

    }
}
```

```
Person in main() after edit
Eric Cartman 10
Number inside main() after edit
13
```

Previous Example: Analysis

- Why did the parameter that was passed by reference change and the simple type (passed by value) did not?

James Tam

Pass By Reference In Java: Important Rules

- **To change the original object (do this all/most of the time)**: Use mutator methods when a reference is passed as a parameter.

```
public void modify(Person aPerson)
{
    aPerson.setName("Eric Cartman");
    aPerson.setAge(10);
    aNum = 888;
}
```

- **Do not**: Use an assignment statement when the reference is passed as a parameter.

```
public void modify(Person aPerson)
{
    aPerson = new Person("Kenny"); //Creates a new object.
}
```

James Tam

Benefits Of Employing References

- References require a bit more complexity but provide several benefits over directly working with objects and arrays.
- **Benefit 1**: Reference parameters allows changes to made inside of methods.
  - As you have just seen a reference contains the address of 'something' (object, array).
  - As long as the address of the object or array is retained changes made inside the method will persist after the method ends.
  - Recall that functions or methods can only return zero or one things (passing out of a function after it ends).
  - Passing by reference (passing into the function just as it starts executing) allows more than one change to persist after the function has ended:

`fun(reference1,reference2,reference3…etc.)`

James Tam

Benefits Of Employing References (2)

- **Benefit 2**: If an array or object is large then it's more memory efficient to pass a reference instead.
- Example:
  - References are typically 32 or 64 bits in size.
  - An array or object will almost always be larger.

```
char [] array1 = new char[1000000]; // 4 MB

class SocialNetworkUser
{
    // attribute for images
    // attribute for videos
}
```

James Tam

New Terminology/Definitions

- Shallow and deep copy
- Automatic garbage collection
- Memory leak
- Parameter passing: Pass by value, pass by reference

James Tam

After This Section You Should Now Know

- References
  - How references and objects are related
  - The difference between a deep vs. shallow copy
  - What is the difference between comparing references vs. objects
  - What is automatic garbage collection and how it's related to the use of references
- How the two methods of parameter passing work, what types are passed using each mechanism
- What are the benefits of employing references

James Tam

Copyright Notification

- “Unless otherwise indicated, all images in this presentation are used with permission from Microsoft.”