Advanced Java Programming

Part 3: wrapper classes, arrays of 'objects'

James Tam

Modifying Simple Types (Parameters)

- What to do when only one thing needs to be changed: return the updated value after the method ends
- What to do when more than one thing needs to be changed:
  - Pass an array (e.g., three integers must be modified in a method, then pass an array of integers with 3 elements).
  - Enlist the aid of a wrapper (class).

Image copyright unknown

James Tam

Wrapper Classes

- A class definition built around a simple type

```
public class Coordinate {
    private int xCoordinate;
    private int yCoordinate;
    ...
}
```

- Benefits illustrated by this example:
  - Related pieces of information can be passed into methods together rather than separately.

```
Coordinate aLocation = new Coordinate();
Method(aLocation);   // vs method(x,y);
```

  - The values of two atomic types x & y can be changed inside a method call (because an object 'wraps' them and the object is passed by reference).

James Tam

Wrapper Classes (2)

- Also Wrapper classes are also used to provide class-like capabilities (i.e., methods) to simple types (e.g., `int`) e.g., class `Integer`
  - https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/lang/class-use/Integer.html (last accessed Feb 2021)
  - Example useful method `parseInt(String)`: converting strings to integers

```
int num = Integer.parseInt("123");  // More on this later
```

James Tam

Arrays: Parameters And Return Values

- **Name of the folder containing the complete example** : `9arrayParameters`

- **Format, method call**:
  -When the method is called, passing an array as a parameter and storing a return value appears no different as passing other types.
  -**Example** (`list1` and `list2` are arrays)

```
list2 = ape.oneDimensional(list1);
```

James Tam

Arrays: Parameters And Return Values (2)

- **Format, method definition**:
  - Use 'square brackets' to indicate that the return value or parameter is an array.
  - Each dimension requires an additional square bracket.
  - One dimensional:

```
public int [] oneDimensional(int [] array1) { ... }
```

  - Two dimensional:

```
public char [][] twoDimensional(char [][] array1) {
    ...
}
```

James Tam

Array Of 'Objects'

- Although referred to as an array of objects they are actually arrays of references to objects.
- Recall for arrays: 2 steps are involved to create the array

```
int [] array;          // Reference to array
array = new int[3];    // Creates array of integers
```

- Recall for objects: 2 steps are required to create the object

```
Person jim;          // Reference to Person object
jim = new Person();  // Creates object
```

James Tam

Array Of 'Objects' (2)

- An array of objects is actually an array of references to objects.
- So 3 steps are usually required
  - Two steps are still needed to create the array

```
// Step 1: create reference to array
Person [] somePeople;

// Step 2: create array
somePeople = new Person[3];
```

  - In Java after these two steps each array element will be null.

```
somePeople[0].setAge(10);  // Null pointer exception
```

James Tam

Array Of 'Objects' (3)

- The third step requires traversal through array elements (as needed): create a new object and have the array element refer to that object.

```
for (i = 0; i < 3; i++)
{
    // Create object, array element refers to that object
    somePeople[i] = new Person();

    // Now that array element refers to an object, a method
    // can be called.
    somePeople[i].setAge(i);
}
```

James Tam

Array Of Objects: Example

- **Name of the folders containing the complete example** : `9arrayOfReferences/simple`

James Tam

Class Person

```
public class Person {
    private int age;

    public Person() {
        age = 0;
    }

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

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

James Tam

Driver Class

```
public class Driver
{
    public static void main(String [] args) {
        Person [] somePeople; // Reference to array
        int i;
        somePeople = new Person[3];  // Create array
        for (i = 0; i < 3; i++) {
            // Create object, each element refers to a newly
            // created object
            somePeople[i] = new Person();
            somePeople[i].setAge(i);
            System.out.println("Age: " +
                               somePeople[i].getAge());
        }
    }
}
```

```
Age: 0
Age: 1
Age: 2
```

James Tam

Design Example

- Suppose we wanted to simulate a 2D universe in the form of a numbered grid ('World')

```
class World
{
    private [][] Tardis grid;
}
```

- Each cell in the grid was either an empty void or contained the object that traveled the grid ('Tardis')[1]

```
class Tardis
{

}
```

1 Tardis and "Doctor Who" © BBC

James Tam

General Description Of Program

- The 'world/universe' is largely empty.
- Only one cell contains the Tardis.
- The Tardis can randomly move from cell to cell in the grid.
- Each movement of Tardis uses up one unit of energy.

James Tam

Designing The World

Class World

- Attributes?
- Methods?

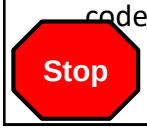
Class Tardis

- Attributes?
- Methods?

James Tam

CAUTION: STOP READING AHEAD

- JT's note: Normally you are supposed to read ahead so you are prepared for class.
- In this case you will get more out of the design exercise if you don't read ahead and see the answer beforehand.
- That will force you to actually think about the problem yourself (and hopefully get better at designing your own programs).
- So for now skip reading the slides that follow this one up to the one that has a corresponding 'go' symbol all over it.
- After we have completed the design exercise in class you should go back and look through those slides (and the source code).



James Tam

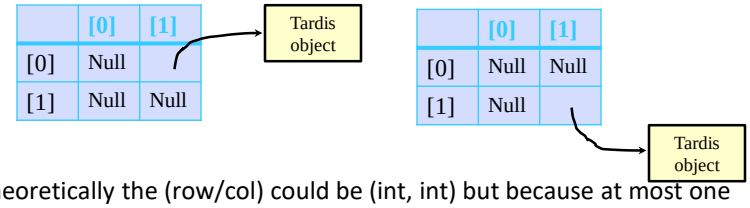
Tardis

- Attributes
  - Current energy level
- Methods:
  - Randomly generating movement:
    - Some method must reduce the energy level as the Tardis moves
    - The actual 'movement' from square to square in the grid will be a responsibility of class `World` because the grid is an attribute of the world.

James Tam

World

- Attributes
  - A 2D array that stores information about the 'universe'
  - Most array elements will be empty (`null`)
  - One element will refer to the Tardis object
  - The maximum number of rows and columns
  - The current location (row/column) of the Tardis
    - Needed to 'move' the Tardis from source cell to destination cell



  - Theoretically the (row/col) could be (int, int) but because at most one item can be returned from a method the location will be tracked as a `Location` object (details in code):
    - `World.move()->Tardis.calculateCoordinates()`

James Tam

World (2)

- Methods
  - Constructor(s) to create the world
  - Methods that modify the world (e.g., making sure each array element is truly null: `wipe()`
  - Displaying the world: `display()`
  - Changing the contents of the objects in the world (e.g., editing the world or moving objects): `move()`

James Tam

Manager

- It is responsible for things like determining how long the simulation runs.
- For very simple programs it may be a part of the `World` class (in this case it's part of the `Driver`).
- But more complex programs (e.g., need to track many pieces of information like multiple players, current scores etc. and simulation rules) may require a separate `Manager` class.
  - The `Driver` will then likely be responsible for instantiating a `Manager` object and calling some method of the manager to start the simulation.

James Tam

GO! GO!

END SECTION: Proceed Reading

- You can continue reading ahead to the slides that follow this one.
  - JT: Thank you for your understanding and co-operation.

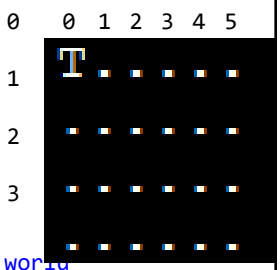
GO! GO!

Source Code: Design Exercise

- **Name of the folder containing the complete example** : `9arrayReferences/doctor`

James Tam

Class Tardis




```
public class Tardis {
    private int energy;
    public Tardis(int startEnergy) {
        energy = startEnergy;
    }
    // max row and column define the size of the world
    public Location calculateCoordinates(int maxRow,int maxColumn){
        Random aGenerator = new Random();
        Location aLocation = new Location();
        aLocation.setRow(aGenerator.nextInt(maxRow));
        aLocation.setColumn(aGenerator.nextInt(maxColumn));
        System.out.println("Tardis rematerializing at (r/c): " +
          aLocation.getRow() + "/" + aLocation.getColumn());
        energy--;
        return(aLocation);
    }
```

e.g., = 4 0, 1, 2, 3

e.g., = 7 0, 1, 2, 3, 4, 5, 6

James Tam

Class Tardis (2)

```
    public boolean hasEnergy()
    {
        if (energy > 0)
            return(true);
        else
            return(false);
    }
}
```

James Tam

Class Location

```
public class Location
{
    private int row;
    private int column;

    public Location() {
        row = 0;
        column = 0;
    }

    public Location(int aRow, int aColumn) {
        row = aRow;
        column = aColumn;
    }
```

James Tam

Class Location (2)

```
    public int getColumn() {
        return(column);
    }
    public int getRow() {
        return(row);
    }
    public void setColumn(int aColumn) {
        column = aColumn;
    }
    public void setRow(int aRow){
        row = aRow;
    }
}
```

James Tam

Class World: Attributes

```
public class World
{
    private Tardis [][] grid;  // Simulated world
    private int maxRow;     // Row capacity
    private int maxColumn;  // Column capacity
    private Location tardisLocation;   // (row/col) of Tardis
```

James Tam

Class World: Constructor

```
    public World() {
        final int START_ROW = 0;
        final int START_COLUMN = 0;
        Scanner in = new Scanner(System.in);
        System.out.print("Max rows: ");
        maxRow = in.nextInt();
        System.out.print("Max columns: ");
        maxColumn = in.nextInt();
        grid = new Tardis[maxRow][maxColumn];
        wipe();   // Empties the world
        // Put the Doctor's Tardis here.
        grid[START_ROW][START_COLUMN] = new Tardis(10);
        tardisLocation = new Location(START_ROW,START_COLUMN);
        display();
}
```

James Tam

Class World: Initialization

```
public void wipe()
{
    int r;
    int c;
    for (r = 0; r < maxRow; r++)
    {
        for (c = 0; c < maxColumn; c++)
        {
            grid[r][c] = null;
        }
    }
}
```

e.g., max = 2

e.g., max = 3

r = 0, c = {0,1,2}

r = 1, c = {0,1,2}

	[0]	[1]	[2]
[0]	null	null	null
[1]	null	null	null

James Tam

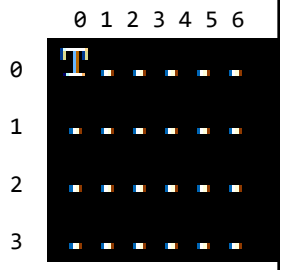
Class World: Display

```
public void display()
{
    int r;
    int c;
    for (r = 0; r < maxRow; r++)
    {
        for (c = 0; c < maxColumn; c++)
        {
            if (grid[r][c] == null)
                System.out.print(".");
            else
                System.out.print("T");
        }
        System.out.println();
    }
}
```

e.g., = 4

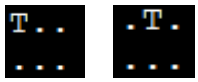
e.g., = 7

Move cursor to display new row on next line



James Tam

Movement

- To make it look like the Tardis has 'moved'. 
- Set the destination (row/column) to refer to the Tardis object.
- Set the source (row/column) to `null`.

Before move

	[0]	[1]
[0]	Null	
[1]	Null	Null

Tardis object

After move

	[0]	[1]
[0]	Null	Null
[1]	Null	

Tardis object

James Tam

Class `World`: Move

```
    public void move()
    {
        int currentRow = tardisLocation.getRow();
        int currentColumn = tardisLocation.getColumn();

        // Keep track of where the Tardis is currently located
        int oldRow = currentRow;
        int oldColumn = currentColumn;

        // (cRow,cCol) location of Tardis in the world grid
tardisLocation =
grid[currentRow][currentColumn].calculateCoordinates
  (maxRow,maxColumn);
```

Recall:
`Tardis.calculateCoordinates()`
randomly generates a new (row/column) and returns a `Location` object

James Tam

Class World: Move (2)

```
// Update temporary values with current location
currentRow = tardisLocation.getRow();
currentColumn = tardisLocation.getColumn();

// Copy tardis from the old location to the new one.
grid[currentRow][currentColumn] = grid[oldRow][oldColumn];

// Check if tardis trying to move onto same square, don't
// 'wipe' if this is the case or tardis will be lost
// (Tardis object becomes a memory leak).
if ((currentRow == oldRow) &&
    (currentColumn == oldColumn)) {
      System.out.println("Same location");
}
else {
      // ‘wipe’ tardis off old location
      grid[oldRow][oldColumn] = null;
}
```

James Tam

Class World: Move (3)

```
    System.out.println("Tardis re-materializing");
    display();
}
```

James Tam

Class `World`: Querying Energy

```
public boolean energyRemains()
{
    boolean isThereEnergy;
    isThereEnergy =
grid[tardisLocation.getRow()][tardisLocation.getColumn()].
hasEnergy();
    return(isThereEnergy);
}
```

James Tam

The `Driver` Class (Also The "Manager")

```
public class Driver
{
    public static void main(String [] args) {
        Scanner in = new Scanner(System.in);
        World aWorld = new World();
        while (aWorld.energyRemains() == true)
        {
            aWorld.move();
            System.out.println("Hit enter to continue");
            in.nextLine();   }
            System.out.println("\n<<<Tardis is out of energy,
              end simulation>>> \n");
        }
    }
}
```

James Tam

After This Section You Should Now Know

- What is a wrapper class and the value provided by using them.
- How to pass arrays as parameters and return them from methods
- Arrays of 'objects'
  - Why they are really arrays of references
  - How to declare such an array, create and access elements
- How to create a simple simulation using an array of references

James Tam

Copyright Notification

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

James Tam