Objects and memory

The stack
The heap
Variables live and die
null
Clone
Serializing objects

Memory: A programmer’s prospective

- Recall: Memory can be viewed as a sequence of numbered ‘storage boxes’
  - Essentially memory is an array
- The compiler decides (on our behalf) where variables will live in memory (their address)
  ```
  public static void main(String[] args) {
    int x = 0;
    char c = 'a';
    double dx = 140000000000;
    ...
  }
  ```
  - End method main

<table>
<thead>
<tr>
<th>Symbol Table</th>
<th>Name</th>
<th>Type</th>
<th>Length</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>int</td>
<td>1</td>
<td>x001</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>char</td>
<td>1</td>
<td>x002</td>
<td></td>
</tr>
<tr>
<td>dx</td>
<td>double</td>
<td>2</td>
<td>x003</td>
<td></td>
</tr>
</tbody>
</table>

- Areas of memory are allocated for the different types of things that need to be stored
  - Operating System routines
  - The executable code
  - Static/global variables
  - Dynamically created objects
    - Instance variables
    - Memory needs grow/shrink during execution of the program.
    - The compiler has no idea how much space you might need (may depend on input, etc)
  - Local variables (a.k.a. stack variables)
    - Memory requirements change as variables local to the method come into and out of scope
- The amount of available memory is system dependent
Types of variables

```java
public class Main {
    public static int number;
    public static void main(String[] args) {
        System.out.println(number);
    }
}

public class Point {
    int x;
    int y;
}
```

Memory in action

```java
public class Point {
    int x;
    int y;
    Point (int xValue, int yValue) {
        x = xValue;
        y = yValue;
    }
    Point makePoint(int x, int y) {
        Point p1 = new Point(x,y);
        return p1;
    }
    static void main(String[] args) {
        Point p1 = new Point(1,2);
        Point p2 = makePoint(3,4);
    }
}
```

After "new" but before constructor is executed
public class Point {
    int x;
    int y;
    Point (int xValue, int yValue) {
        x = xValue;
        y = yValue;
    } // end constructor
    Point makePoint (int x, int y) {
        Point p1 = new Point(x,y);
        return p1;
    } // end method Point
    static void main (String[] args) {
        Point p1 = new Point(1,2);
        Point p2 = makePoint(3,4);
    } end method main
} // end class Point

Memory in action

constructor is about to begin

constructor is about to finish

Execution of the first line of main complete

main():() args
main(): p1
main(): p2
Point(): yValue        2
Point(): xValue        1
public class Point {
    int x;
    int y;
    Point (int xValue, int yValue) {
        x = xValue;
        y = yValue;
    } // end constructor
    Point makePoint (int x, int y) {
        Point p1 = new Point(x, y);
        return p1;
    } // end method Point
    static void main (String[] args) {
        Point p1 = new Point(1, 2);
        Point p2 = makePoint(3, 4);
    } // end method main
} // end class Point

Preparing to execute makePoint

stack
makePoint, after new, before constructor called
makePoint(): x         4
makePoint(): y         3
makePoint(): p1

makePoint near end of constructor call
makePoint(): yValue         4
makePoint(): xValue         3
**Memory in action**

```java
public class Point {
    int x;
    int y;
    Point (int xValue, int yValue) {
        x = xValue;
        y = yValue;
    } // end constructor
    Point makePoint (int x, int y) {
        Point p1 = new Point(x,y);
        return p1;
    } // end method Point
    static void main (String[] args) {
        Point p1 = new Point(1,2);
        Point p2 = makePoint(3,4);
    } end method main
} // end class Point
```
Stack Trace

Exception in thread "main"
java.lang.StringIndexOutOfBoundsException:
String index out of range: 3
at java.lang.String.charAt(Unknown Source)
at Coordinate.getStatus (Coordinate: 67)
at Ship.getLocation (Ship: 15)
at GameBoard.checkTarget (GameBoard.java: 108)
at GameBoard.play (GameBoard.java: 54)
at Main.main (Main.java: 23)

Objects are allocated

* When you create an object instance memory is allocated on the heap.
  - There is NO WAY to create an object other than someone,
    somewhere using new on the class type
  - An object instance needs a place to store its instance variables.
  - The instance variables determine the size of the data structure.
  - heap: Essentially, a list in which data structures can be added or
    removed in any order and at any time.
  - Objects can be allocated anywhere in the heap where there is space
    (not necessarily adjacent to each other)
  - Any number of instances can be created, each starting at a unique
    address/location in the heap.
  - Object reference variables (which could be static, local, or instance) hold
    the reference/location of the object on the heap
  - no live variable with reference <==> object is inaccessible garbage

Static variables have a permanent home

* When you create a static variable, it is assigned a permanent location in
  memory
  - A static variable is always live
    - (from the time the class is loaded at start until the program ends)
  - Static variables don’t exist in the object’s data structure in the heap
    - This is why a static variables has the same value for all instances of a
      class. They all refer to the same memory location
  - You should always assign an initial value to static variables
    - In Java and C++, static variables are assigned a default value of 0
What value should an object reference variable have BEFORE an object is created and stores its reference in the variable?

- null
  - is a keyword
  - can be stored in a reference variable
  - indicates that the variable currently refers to no existing object

The concept of null is particularly important in languages without garbage collection, as references may be stored to objects that have been destroyed.

In general, methods have a precondition that the parameters contain valid object references.
- Best practices imply that one should always test for null references before using an object reference passed as a parameter.

Example: Memory

```java
public class PointList extends Point {
    PointList nextPoint = new PointList();
    static PointList firstPoint;
    PointList () {
        super(0, 0);
    } // end constructor

    public static void main (String[] args) {
        PointList list = new PointList();
        list = null;
    } // end method main
} // end class Point
```

- What is the likely intention of this class?
- What are the static, instance, and local variables?
- This code compiles without error. When executed, you get a runtime exception: StackOverflowError. Why and how?

Aggregate object

- Are cards private or not?
Memory and encapsulation

Deck
  cardsInDeck

ArrayList

Card
  suit:
  rank:

public class Poker {
  public void cheat () {
    Card topCard = topCard();
    topCard.setRank("Ace");
  } // end method cheat
} // end class Poker

Security and encapsulation

- When an accessor method exists to provide access to a private object:
  - Do not return the reference to the actual field object
    - This would allow the external program to access/change the object
      without going through the appropriate mutator method!
  - Instead, return a copy of the object.
    - This provides all the same information
      - But changes to the copy do not affect the original!
- You have to make a new object
- Wouldn’t it be great if we had a constructor that made this easy?

Copy constructors

- Copy constructor: A constructor that accepts an object of the same class
  as an argument and makes an “identical” copy.

public class Deck {
  private ArrayList<Card> cardsInDeck;
  public Card topCard() {
    // end method topCard
  }
} // end class Deck

public class Card {
  public Card (Card originalCard) {
    this.setSuit(originalCard.getSuit());
    this.setRank(originalCard.getRank());
  } // end copy constructor
} // end class Card

Why are Strings Immutable in Java?
Shallow Copy

```java
public class Deck {
    private ArrayList<Card> cardsInDeck;
    // ... 
    public Deck (Deck originalDeck) {
        this.cardsInDeck = new ArrayList<Card>();// Initialize a new ArrayList
        for (Card card : cardsInDeck) {// Add the cards
            this.cardsInDeck.add(card);
        }
    } // end (SHALLOW) copy constructor
    // ... 
}
```