OOP: Objects

Advanced Programming

Objects

- Objects are instances of a class
- Objects are created at run time and the memory in the heap

Object creation

- Instantiating an object means creating an object of a given class in the Heap. In Java, objects are created using the keyword "new"
  - CoffeeMaker aCoffeeMaker = new …
- Initialization: the new operator is followed by a call to a constructor (in the stack) , which initializes the new object (initially to Null is otherwise stated, see next)

CoffeeMaker aCoffeeMaker = new CoffeeMaker();

Reference variables

- To retrieve an object of a class use object reference variables (references to objects)

CoffeeMaker aCoffeeMaker

- (Object) reference variable declaration: associates a variable name with a class

Default values in Java

- Every variable must have a value before its value is used
- Each variable has default value when it is created
  - For all reference variables the default value is Null
  - for primitive variables (int, short, char byte, long, float and double, boolean) is zero or char (\u0000) or boolean (false)

Reference to null

- In Java: setting a reference to Null or not initializing it:
  - If an application calls an object through a reference not yet initialized or null, the Java compiler will return an error message of the type NullPointerException

Example of object instantiation

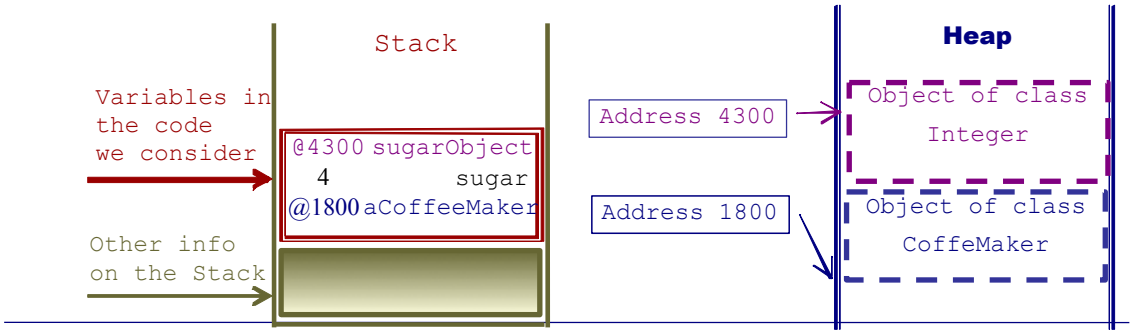
```
CoffeeMaker aCoffeeMaker = new CoffeeMaker();
```

Accessing objects in Java

- A reference variable holds the address of an object in the heap

Example

- CoffeeMaker aCoffeeMaker;
- aCoffeeMaker = new CoffeeMaker();
- int sugar = 4;
- Integer sugarObject = new Integer(3);



Variables and References

- Stack variables are all created on the Stack statically
  - e.g., object reference variables or primitive variables in methods
- Instance variables are created on the Heap dynamically
  - e.g., object reference variables in classes

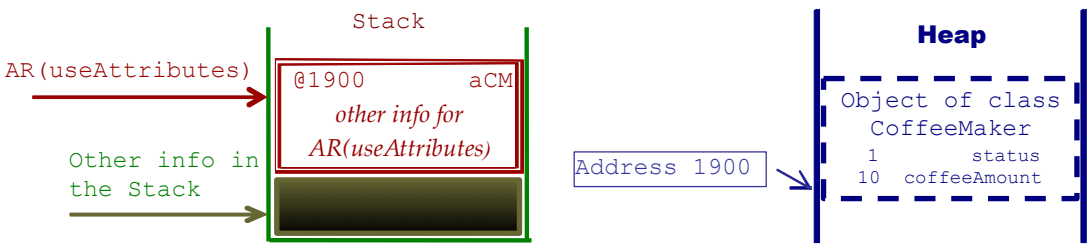
Object population

- In the heap objects are populated with the values for the instance variables

Example

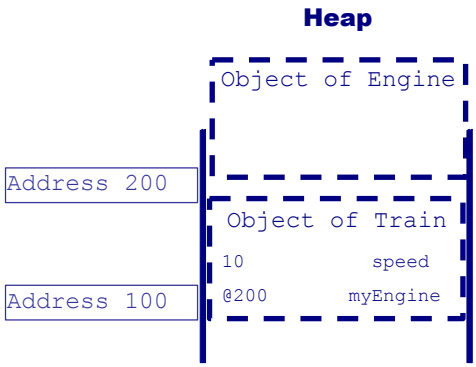
AR of constructors are omitted in this example

```
void useAttributes() {
  CoffeeMaker aCM = new CoffeeMaker();
  aCM.status = 1;
  aCM.coffeeAmount = 10;
}
```



Instance Variables

```
public class Train{
   private speed = 10;
   Engine myEngine = new Engine();
}
public class Engine{}
Train myTrain = new Train();
```



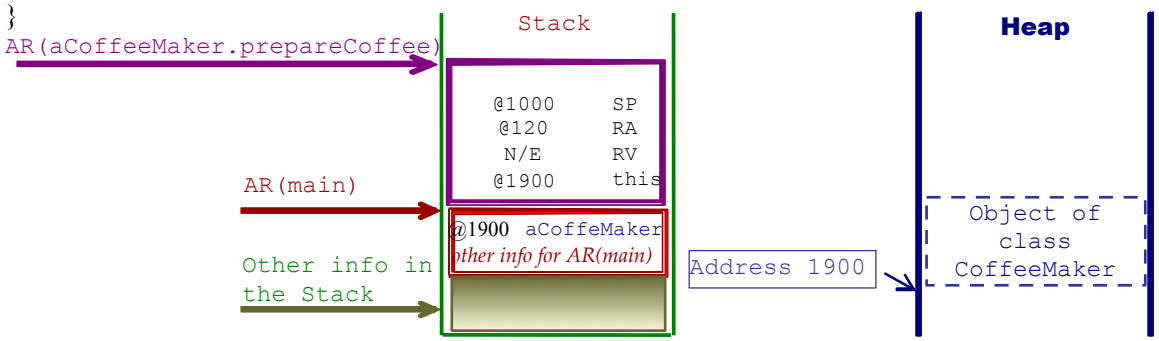
Methods

- The task of a program is accomplished via the interaction of objects
  - The interaction is based on the exchange of messages
  - Upon reception of a message, an object replies with a predetermined method
  - Therefore, we can have an object of class Dog, Pluto, and an object of class People, John. When Chunk sends Rea the message sit, Rea executes its method associated with the message sit.

Instance Methods

```
public class CoffeeMaker {
  public void prepareCoffee() {
}
  public void prepareCoffeeSweet(int sugarAm){
}
  void main(...) {
    CoffeeMaker aCoffeeMaker;
    aCoffeeMaker = new CoffeeMaker();
    aCoffeeMaker.prepareCoffee();
  }
}
```

AR of constructors are omitted in this example



Instance methods

aCoffeeMaker.prepareCoffee();

- From a memory model standpoint, invoking a method from an object is quite like calling a function, but there is the need for tracing which object has invoked the method
  - in the AR of a method we add **`this`** referencing the object address in the heap
- In the stack, the AR of an instance method is labelled with AR(referenceVariable.methodName)

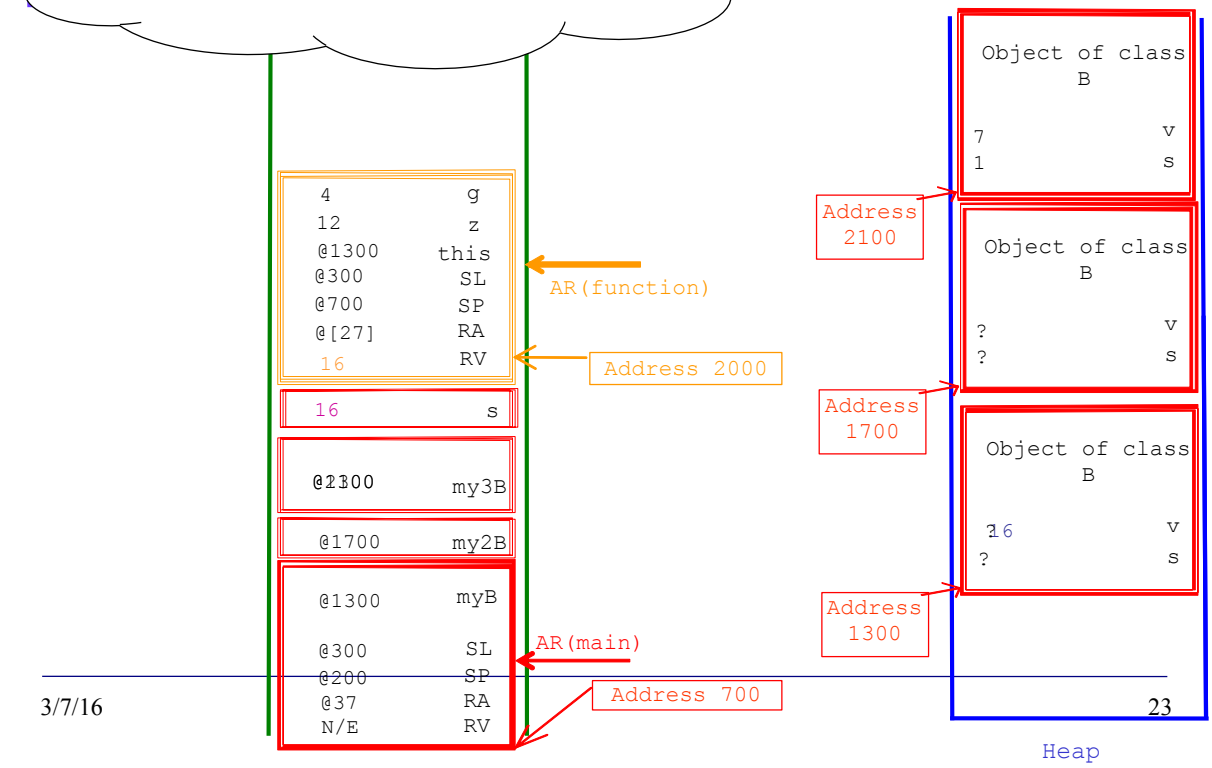
Source Code

```
[1] public class B {
[2]    int v;
[3]    int s;
[4]    public B(){
[5]         int k = 0;
[6]         int m = function(k);
[7]    }
[8]
[9]    public B (int z){
[10]          int s = 34;
[11]   }
[12]   public B (double d){
[13]          s = (int) d;
[14]          v = (int) d+6;
[15]   }
[16]
[17]   private int function(int z){
[18]         int g = 4;
[19]         return  g+z;
[20]   }
```

```
[21]    public static void main (…){
[22]         B myB = new B();
[23]         B my2B = new B(3);
[24]         B my3B = new B(1.0);
[25]         my3B = myB;
[26]         int s = myB.function(12);
[27]         myB.v = s;
[28]    }
[29]}
```

There are several instantiations of the class B. "this" in the AR(function) points to the address of the correct object in the heap

Constructors are omitted
How many time "function" is called?
Which object will be cleaned up by the garbage collector?
"this" is missing. Where do you put it and what is its value?



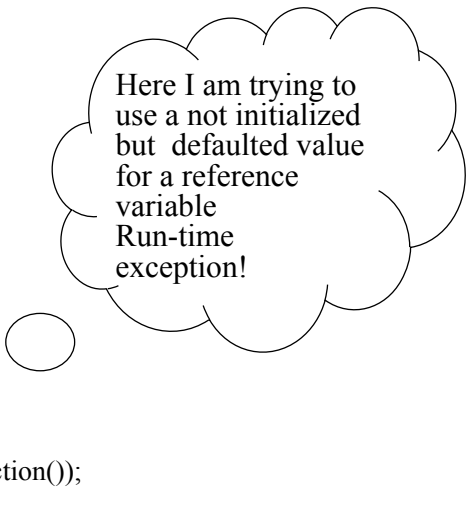
Instance methods

aTrain.move(<parameter_list>)

- calls the method **move** with appropriate actual arguments <parameter_list> for the object referenced by the reference variable **aTrain**
- move() acts on the object referenced by aTrain

Do not use non-initialized values!

```
public class Track{
   private int value;
   public Track(int v) { value = v; }
   public int getDirection() { return value; }
}
 public class Train{
   private Track theTrack;
   public static void main(String[] args) {
      Train myTrain = new Train();
      System.out.println(myTrain.theTrack.getDirection());
   }
 }
```



Memory management in the Heap

- Objects with no references pointing to them are considered eligible for automatic garbage collection by the system
- The garbage collector runs periodically and performs the real destruction of these objects
- Developers need not to worry about memory release

Memory management in the Heap

- When the null value is assigned to a reference, the previously referenced object will be released and if not used anymore destroyed releasing the memory
- Thus explicit object destruction is never an issue in Java (except in Java Native Interface and connection to database)
- Garbage collection is not directly under control of the programmer, hence problems could arise if strictly predictable timing behavior is needed (as in real-time systems)

Memory diagram