Review of references

- References point to objects

- A reference points to an instance of a particular class

- Declare a reference
  
  Integer x;
Review of objects

- Classes define objects

- An object is an *instance* of a particular class

- Invoke a constructor to create an object:
  
  ```java
  new Integer(3);
  ```
Review of assignment

```java
public class AssignmentReview {
    public static void main(String[] args) {
        Integer num;
        num = new Integer(3);
        Integer x = num;
        Integer y = new Integer(3);
        Integer z;
    }
}
```
Introducing the Java Heap

The Java Heap shows what references and objects exist at runtime:

num \xrightarrow{x} Integer: 3

y \xrightarrow{} Integer: 3

z \xrightarrow{} null
Null references

- Unassigned references point to `null`

- `null` is not an object (no fields, no methods)

- `z.intValue()` results in an error
  - (a `NullPointerException`, to be exact)
Assignment versus mutation

- Use “=” to assign an object to a reference
- Some methods mutate their objects
- References may share objects, so beware of side effects
Mutation of shared object

public class MutationExample {
    public static void main(String[] args) {
        List<String> a = new ArrayList<String>();
        List<String> b = a; // b & a share the List
        a.add(“Hello, world!”);
        System.out.println(b);
        // Prints “Hello, world!”
    }
}
Mutation of shared object

Java Heap:

List<String>:
[“Hello, world!”]
Static versus non-static

- Fields and methods may be declared “static”
- Static members belong to the class
- Non-static members belong to instances of the class
Non-static fields

public class Bean {
    public int beanCounter = 0;
    public Bean() {
        beanCounter++;
    }
}

public static void main(String[] args) {
    new Bean(); new Bean();
    Bean bean = new Bean();
    System.out.println(bean.beanCounter);
    // Prints "1"
}
Static fields

```java
public class Bean {
    public static int beanCounter = 0;
    public Bean() {
        beanCounter++;
    }

    public static void main(String[] args) {
        new Bean(); new Bean(); new Bean();
        System.out.println(Bean.beanCounter);
        // Prints “3”
    }
}
```
Non-static methods

public class Bean {
    private boolean planted = false;
    public void plantBean() {
        planted = true;
    }
}

public static void main(String[] args) {
    Bean bean = new Bean();
    bean.plantBean(); // Invoked on instance
}
}
static methods

public class Bean {
    private boolean planted = false;
    public static void plantBean(Bean bean) {
        bean.planted = true;
    }

    public static void main(String[] args) {
        Bean bean = new Bean();
        Bean.plantBean(bean); // Invoked on class
        // “bean.plantBean(bean);” legal but inadvisable!
    }
}
Objects passed by reference

```java
public static <T> void removeFirst(List<T> list) {
    list.remove(0);
}

public static void main(String[] args) {
    List<String> myList = new ArrayList<String>();
    myList.add("Cat"); myList.add("Dog");
    removeFirst(myList);
    System.out.println(myList);  // Prints "[Dog]"
}
```
Objects passed by reference

Java Heap:

myList

List<String>:
[“Cat”, “Dog”]

list
References have scope

- Curly braces {...} define regions of scope
- References exist from the time they are declared until they “go out of scope”
- Fields may be referenced throughout class
- Parameters may be referenced throughout method
Examples of scope

```java
public class ScopeExample {
    private int globalField;

    public int method(int parameter) {
        int localVar1;
        if (globalField > 0) {
            int x;
        }
        int localVar2;
    }
}
```
More examples of scope

```java
public class ScopeExample {
    private int globalField;

    public int method(int parameter) {
        int globalField;  // Legal, but hides field!
        int localVar;
        if (this.globalField > 0) {  // Accesses field
            int x;
        }
        int localVar;  // Illegal: same scope
    }
}
```
Quick Morals

- Assignment: References merely point to objects; beware of null pointers

- Static: Don’t invoke static methods on instances

- Pass by Reference: Make a *defensive copy* to avoid accidental mutation

- Scope: Minimize the scope of references as much as possible (e.g. don’t make everything global)