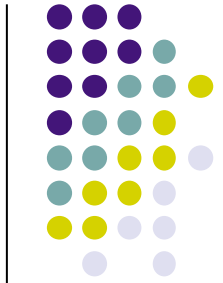
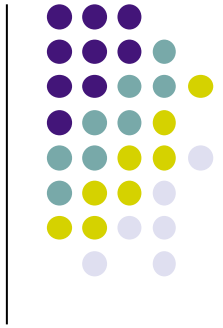
Java semantics with the abstract machine



- As for any language, it is important to understand precisely what the Java language does
  - We can define Java semantics with the abstract machine
  - Most (but not all) of the semantics is straightforward
- We give two examples to show how to give the semantics of Java concepts
  - Parameter passing
  - Static attributes in classes
- For a complete semantics of Java we recommend the book
  - *Java Precisely* by Peter Sestoft, MIT Press, 2005

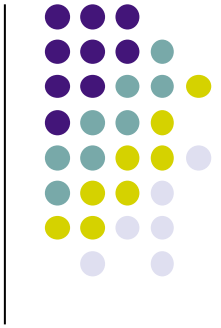
Parameter passing in Java



```
class ByValueExample {
   public static void main(String[] args) {
      double one=1.0;
      System.out.println("before: one = " + one);
      halveIt(one);
      System.out.println("after:  one = " + one);
   }
   public static void halveIt(double arg) {
      arg /= 2.0;
   }
}
```

- Parameter passing is an important part of a language that needs to be understood precisely
- This program calls `halveIt` with argument `one`: what does it print?

Semantics of halveIt

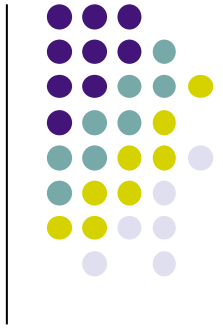


```
public static void halveIt(double arg) {
    arg = arg/2.0;
}
```

```
proc {HalveIt X}
   Arg={NewCell X}
in
   Arg := @Arg / 2.0
end
```

- Here is how to write `halveIt` in Oz
  - This definition gives its semantics
  - This defines only the execution behavior, not the type checking
- The argument Arg is a local cell
  - The number is passed into the local cell
  - Assignments to Arg affect only the local cell, not the cell in the method main
- The number is passed by value

Passing an object parameter



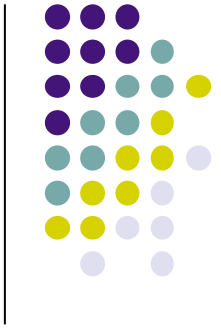
```
class Body {
   public long idNum;
   public String name = "<unnamed>";
   public Body orbits = null;
   private static long nextID = 0;

   Body(String bName, Body orbArd) {
      idNum = nextID++;
      name = bName;
      orbits = orbArd;
   }
}
```

```
class ByValueRef {
   public static void main(String [] args) {
     Body sirius = new Body("Sirius", null);
     System.out.println("bef:"+sirius.name);
     commonName(sirius);
     System.out.println("aft:"+sirius.name);
   }
   public static void commonName(Body bRef) {
     bRef.name = "Dog Star";
     bRef = null;
   }
}
```

- The class `Body` has a constructor (the method `Body`) and a static attribute (the integer `nextID`)
- The program calls `commonName` with the object `sirius`
- The content of `sirius` is modified by `commonName`, but assigning `bRef` to `null` has no effect on `sirius`!

Semantics of commonName

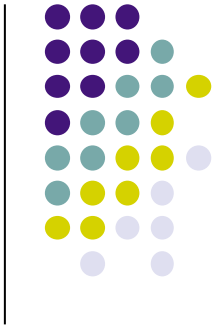


```
public static void commonName(Body bRef)
{
    bRef.name = "Dog Star";
    bRef = null;
}
```

```
proc {CommonName X}
   BRef={NewCell X}
in
   {@BRef setName("Dog Star")}
   BRef:=null
end
```

- Here is how to write `commonName` in Oz
- BRef is a local cell whose content is an object reference
- When CommonName is called, then BRef is initialized with a reference to the object Sirius
- The object reference is passed by value
  - Changes to the content of BRef do not affect the object Sirius

The class Body and its static attribute



```
declare
local NextID Body in
   NextID={NewCell 0}
   class Body
      attr idNum
         name:”<unnamed>”
         orbits:null
      meth initBody(BName OrbArd)
         idNum:=@NextID
         NextID:=@NextID+1
         name:=BName
         orbits:=OrbArd
      end
   end
end
```

```
class Body {
   public long idNum;
   public String name = “<unnamed>”;
   public Body orbits = null;
   private static long nextID = 0;

   Body(String bName, Body orbArd) {
      idNum = nextID++;
      name = bName;
      orbits = orbArd;
   }
}
```

- The definition of class `Body` in Oz gives its semantics
- NextID is a static attribute: a cell defined outside the class, at the same time as the class
  - Not like other attributes which are defined per object
- The constructor `Body` corresponds to method initBody