

Data Types

Primitive types

Whole Numbers: byte (8), short (16), int (32), long (64)

Real Numbers: float (32), double (64)

Other Types: char (16), boolean

```
int count = 0;  
float velocity = 3.4F;  
final double PI = 3.14156D;  
boolean isOn = true, done = false;  
  
velocity = 4.5F;
```

literal constants

initialization

assignment

References to objects in the heap

```
Matrix subMatrix1 = new Matrix();  
Matrix subMatrix2 = subMatrix1;
```

object in dynamic memory (heap)

subMatrix1

subMatrix2

:Matrix

```
Vector velVector = null;
```

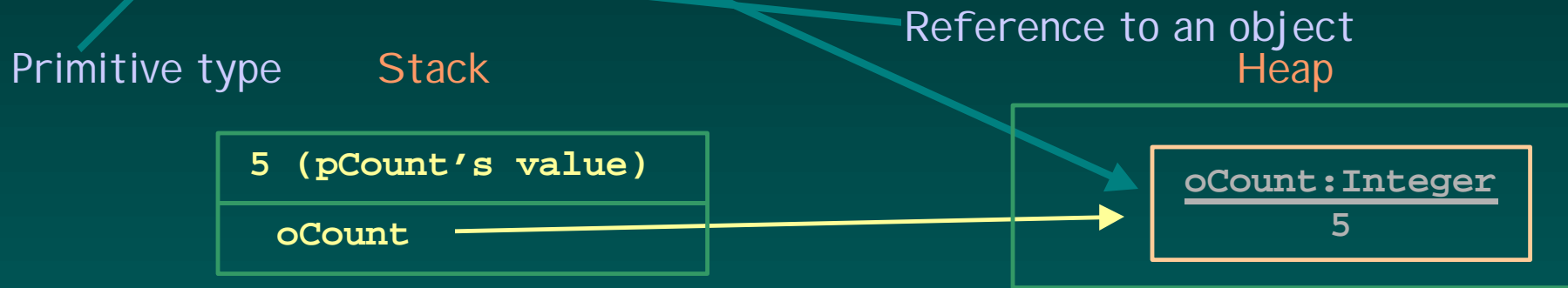
```
final Clock systemClock = new Clock(0.0D, 0.0D, 0.0D);
```

constant reference to an object (reference is const not the object)

Wrappers for primitive types

Wrapper classes can convert Primitive types to Objects

```
int pCount = 5;  
Integer oCount = new Integer(pCount);
```



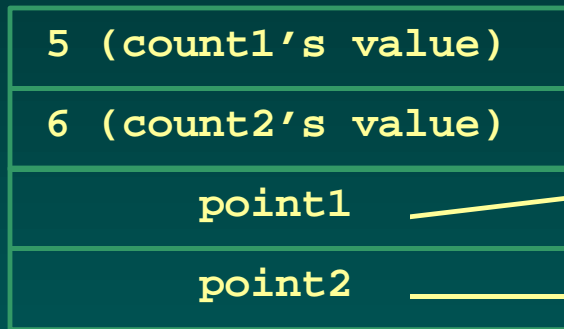
Primitive type	Wrapper Class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Differences between primitive types and references

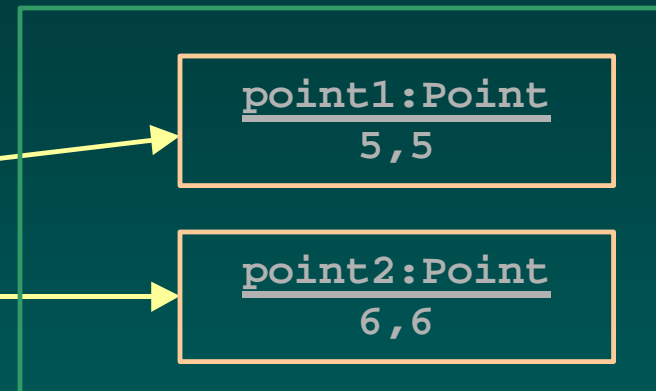
```
int count1 = 5;  
int count2 = 6;
```

```
Point point1 = new Point(5,5);  
Point point2 = new Point(6,6);
```

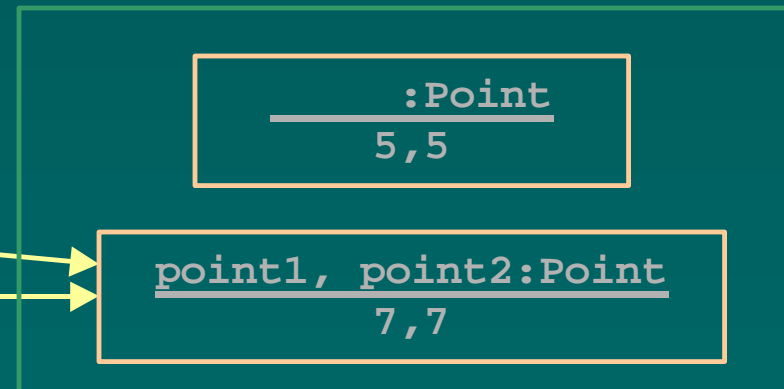
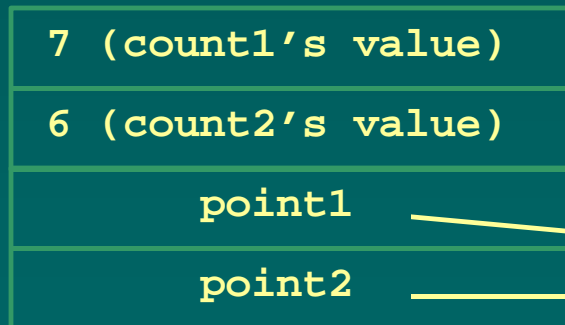
Stack



Heap



```
count1 = count2; count1 = 7;  
point1 = point2; point1.set(7,7);
```



Classes

```
package intro;
```

```
/* This is a comment which  
* spans many lines ..  
* comment ends here */
```

```
public class IntroClass {
```

```
    public static final double PI = 3.14156D;
```

```
    private static float acceleration = 0.0F;
```

```
    private int count = 0;
```

```
    private Velocity finalVel = null;
```

```
    // one line comment goes here ..
```

```
    public void printResult() {
```

```
        double value = Math.sqrt(6.5D);
```

```
        System.out.println( "result is: " + value );
```

```
    }
```

```
    public static Clock sysClock() {
```

```
        Clock autoClock = new Clock(0.0D, 0.0D, 0.0D);
```

```
        .. .. . ;
```

```
        return autoClock;
```

```
    }
```

```
// continued .. ..
```

named constant

static members
(class variable)
one copy per class

instance variables
each instance of IntroClass
has its own copy

local (auto) variables

static method (class method)
can not access instance
variables

classes, continued ...

```
.. .. .
public void setVelocity(Velocity inVel, int stNumber)
{
    finalVel = inVel;
}
public Velocity[] getVelocities() {
    .. .. . ;
}
} // end of class IntroClass
----- New source file -----
package app;

public Class Application {
    public static void main( String[] args) {
        double dRef = IntroClass.PI*56.7D;
        Clock localClock = IntroClass.sysClock();

        IntroClass iClass = new IntroClass();
        iClass.printResult();
    }
} // end of class Application
```

parameters passed by value
(copy of parameter is passed
to the method)

array (knows its length)
`int[] iArray = new int[10];`
`.. "length is "+ iArray.length;`

command line
`% java app.Application local 5.6`
invokes this main method

Command line args

Accessing class
variables and methods
through the class name,
not reference to an
instance

instance methods need references
to instances to be invoked

Call by value

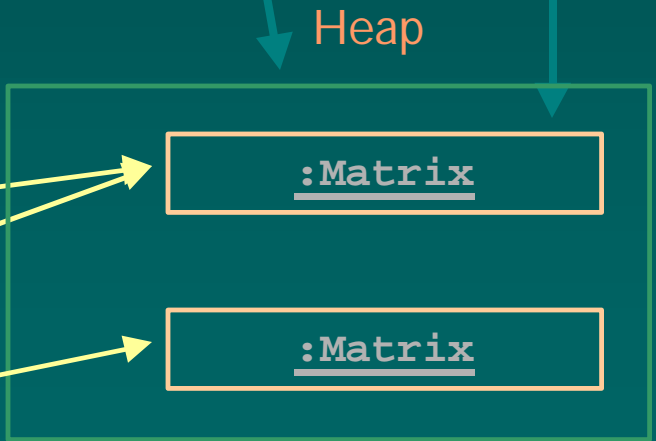
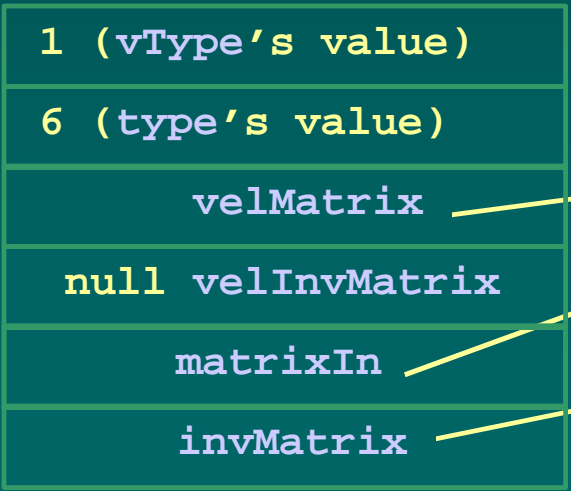
```
public Matrix invert(Matrix matrixIn, int type)
{
    type = 6;
    matrixIn.compact();

    Matrix invMatrix = new Matrix();
    .. .. .
    return invMatrix;
}
```

operation signature
operation prototype
method body

```
Matrix velMatrix = new Matrix();
.. .. .
Matrix velInvMatrix = null;
int vType = 1;
velInvMatrix = invert(velMatrix, vType);
```

snapshot of memory just before the method "invert" exits
method "invert" modifies this object



Stack

Heap

call by value, continued ...

operation signature

operation prototype

```
public Matrix invert(Matrix matrixIn, int type)
{
  type = 6;
  matrixIn.compact();

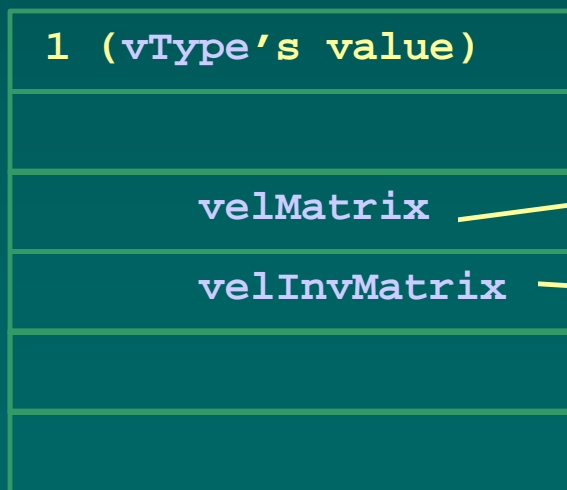
  Matrix invMatrix = new Matrix();
  ... ..
  return invMatrix;
}
```

method body

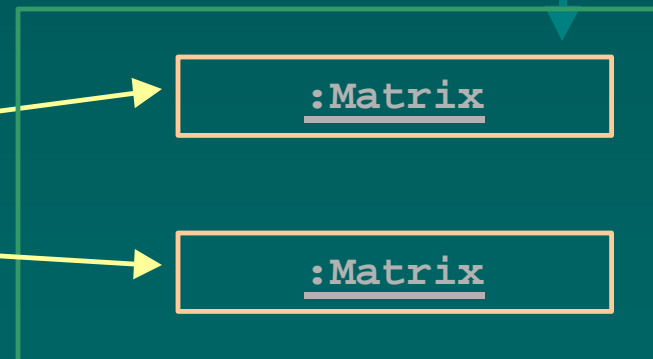
snapshot of memory just after the method "invert" exits

```
Matrix velMatrix = new Matrix();
... ..
Matrix velInvMatrix = null;
int vType = 1;
velInvMatrix = invert(velMatrix, vType);
```

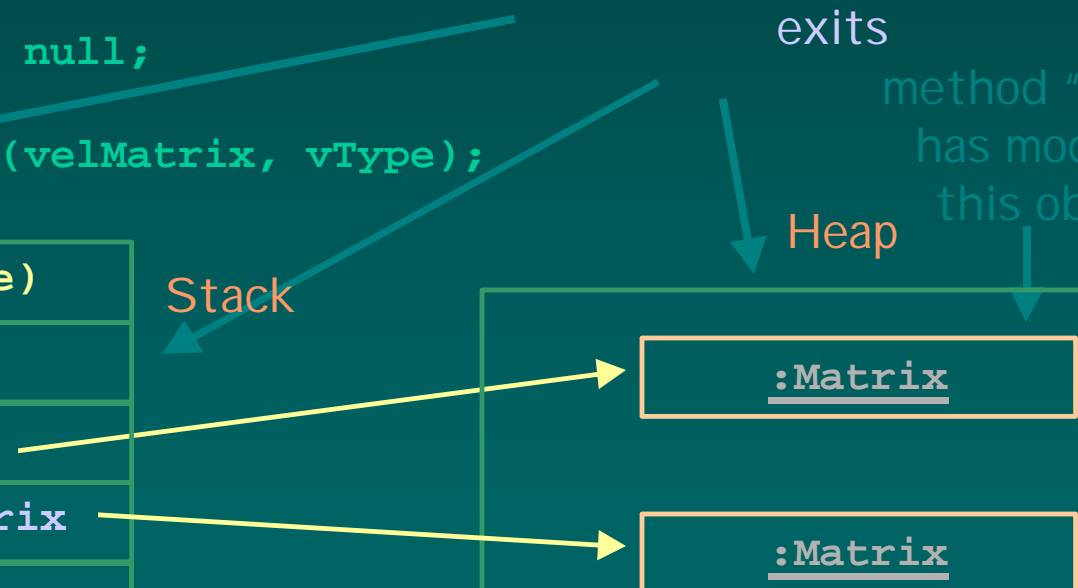
method "invert" has modified this object



Stack



Heap



Math Methods

static methods in `java.lang.Math` class

`java.lang` is a package (no need to explicitly import this package)

Operation prototype

```
double Math.abs(double)
```

```
double Math.sqrt(double)
```

```
double Math.acos(double)
```

```
double Math.asin(double)
```

```
double Math.atan(double)
```

```
double Math.cos(double)
```

```
double Math.sin(double)
```

```
double Math.tan(double)
```

```
double Math.random()
```

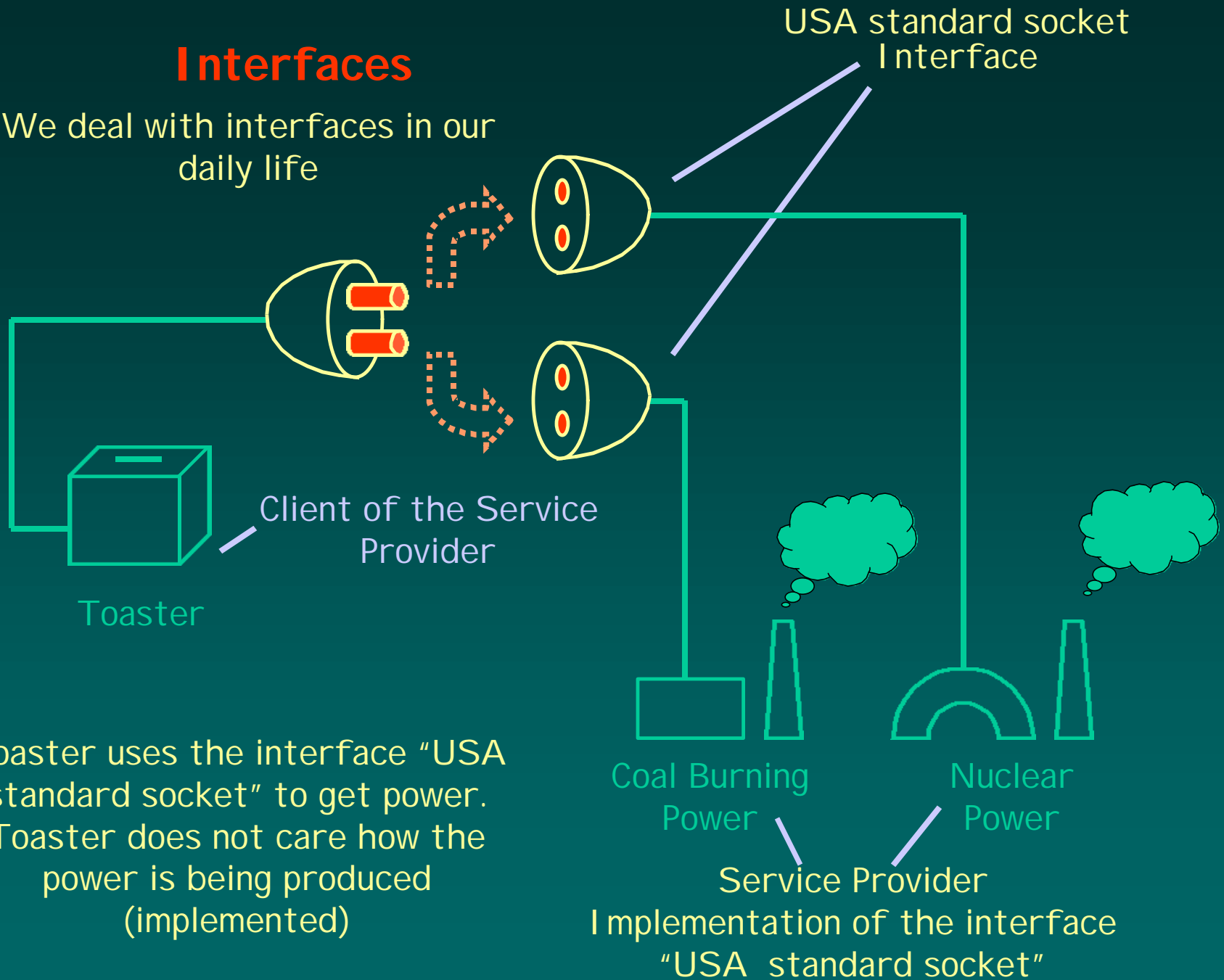
```
double Math.pow(double x, double y) x to power y
```

```
double Math.exp(double)
```

```
double Math.log(double x) natural log ( $\log_e x$ ,  $\ln(x)$ )
```

Interfaces

We deal with interfaces in our daily life



Toaster uses the interface "USA standard socket" to get power. Toaster does not care how the power is being produced (implemented)

Coal Burning Power
Nuclear Power
Service Provider
Implementation of the interface
"USA standard socket"

service



implements

```
public interface USAsocket {
    Power getPower();
}
```



Coal

Nuclear

```
public class CoalPower
implements USAsocket{
    Power getPower(){
        // coal implementation
    }
}
```

```
public class NuclearPower
implements USAsocket {
    Power getPower(){
        // nuclear implementation
    }
}
```

Consumer of service

Toaster knows nothing about coal or nuclear power

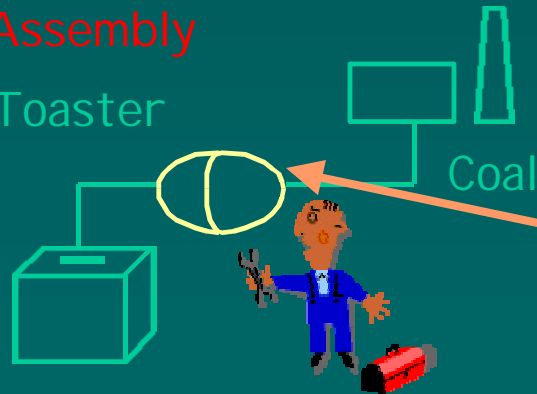


Toaster

```
public class Toaster {
    private USAsocket mySocket = null;
    public void plugIn(USAsocket soc) {
        mySocket = soc;
    }
}
```

Assembly

Toaster



```
public class Assembler {
    // instantiate a new Coal Power plant
    USAsocket mySocket = new CoalPower();
    // buys a new Toaster
    Toaster myToaster = new Toaster();
    // Plugs the Toaster into the socket
    myToaster.plugIn(mySocket);
    ...
}
```

"pseudo" UML

```
public class Toaster {  
    private USAsocket mySocket = null;  
    public void plugIn(USAsocket soc) {  
        mySocket = soc;  
    }  
}
```

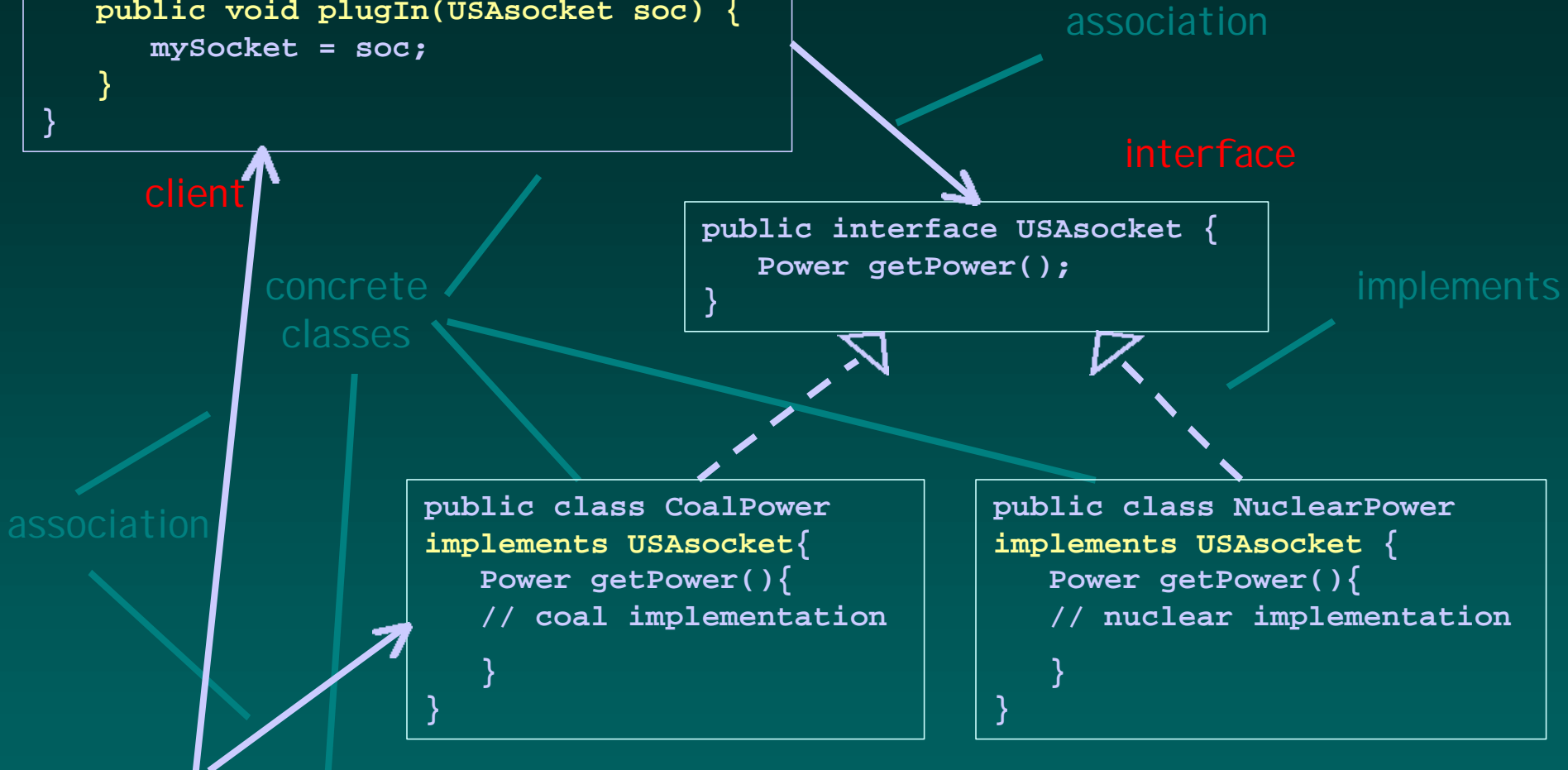
```
public interface USAsocket {  
    Power getPower();  
}
```

```
public class CoalPower  
implements USAsocket{  
    Power getPower(){  
        // coal implementation  
    }  
}
```

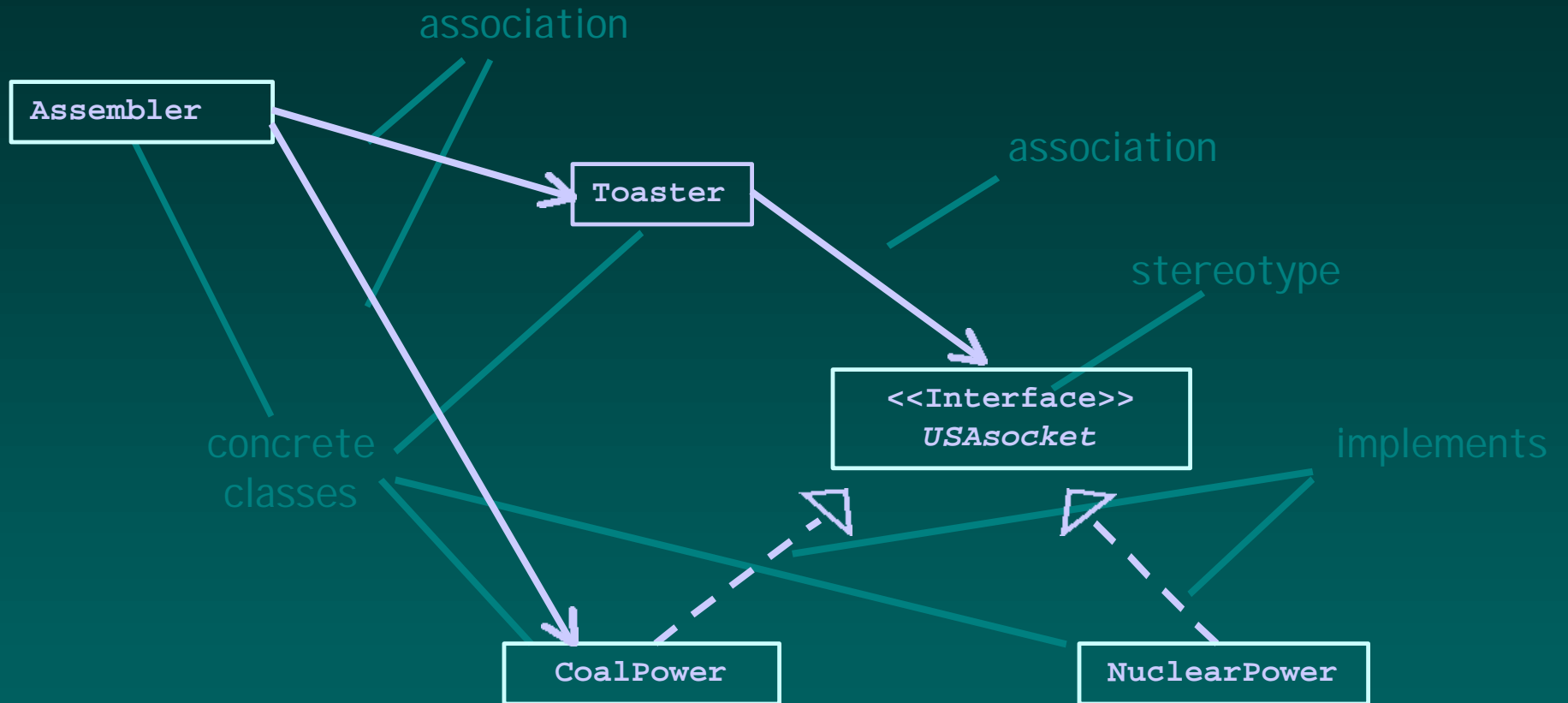
```
public class NuclearPower  
implements USAsocket {  
    Power getPower(){  
        // nuclear implementation  
    }  
}
```

```
public class Assembler {  
    USAsocket mySocket = new CoalPower();  
    Toaster myToaster = new Toaster();  
    myToaster.plugIn(mySocket);  
    ...  
}
```

instantiation and assembly



UML



interfaces ... continued

```
public class CircuitSolver {  
    public void solve() {  
        SqMatrix2D m1 = new PaulSqMatrix2D();  
        .. .. .  
        double tr = m1.trace();  
    }  
}
```

uses

```
public interface SqMatrix2D {  
    double trace();  
    SqMatrix2D inverse();  
}
```

implements

```
public class PaulSqMatrix2D  
implements SqMatrix2D{  
    private double a11 = 0.0D;  
    private double a12 = 0.0D;  
    private double a21 = 0.0D;  
    private double a22 = 0.0D;  
  
    public double trace(){  
        // Paul's trace implementation  
    }  
    public SqMatrix2D inverse(){  
        // Paul's inverse implementation  
    }  
}
```

```
public class BruceSqMatrix2D  
implements SqMatrix2D{  
    private double[] a = null;  
    public double trace(){  
        // Bruce's trace implementation  
    }  
    public SqMatrix2D inverse(){  
        // Bruce's inverse implementation  
    }  
}
```