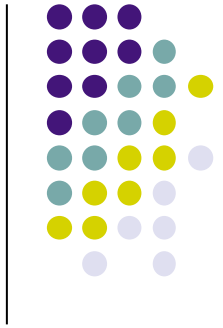
Classes in Java

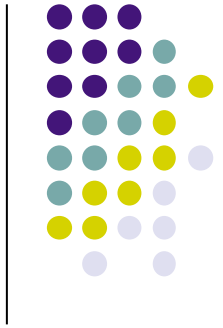


- A Java class has fields (attributes or methods), and members (other classes or interfaces)

- Java has syntax for static and dynamic links
  - The keyword "`super`" gives a static link to the class one level up (as we saw, it should be rarely used!)
  - The keyword "`this`" is used to mean "**self**"

- Java allows single inheritance of classes
  - A class can inherit from exactly one other class

Inheritance example



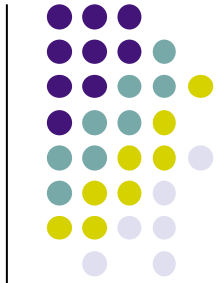
```
class Point {
  public double x, y;

  public void clear() {
     x=0.0;
     y=0.0;
  }
}
```

```
class Pixel extends Point {
  Color color;

  public void clear() {
     super.clear();
     color=null;
  }
}
```

The class Object

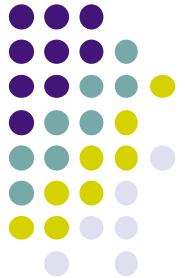


- The class Object is the root of the hierarchy
  - All classes inherit from Object

```
Object oref = new Pixel();
oref = “Some String”;
oref = “Another String”;
```

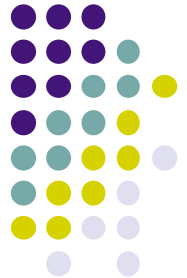
- The reference `oref` can refer to any object
  - We regain some of the flexibility of dynamic typing
  - (String objects are immutable)

Abstract classes and concrete classes



- An abstract class is a class that does not implement all its methods (bodies are missing)
  - An abstract class cannot be instantiated
- A concrete class implements all its methods
  - A concrete class can inherit from an abstract class
  - A concrete class can be instantiated
- With abstract classes, we can write generic programs
  - We define the missing methods using inheritance, to get a concrete class that we can instantiate and execute

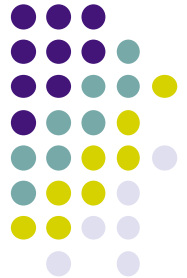
Example of an abstract class



```
abstract class Benchmark {
    abstract void benchmark();

    public long repeat(int count) {
        long start=System.currentTimeMillis();
        for (int i=0; i<count; i++)
            benchmark();
        return (System.currentTimeMillis()-start);
    }
}
```

Doing the same with a higher-order function

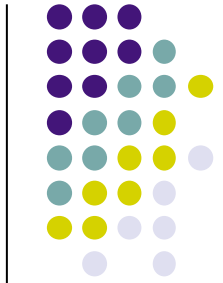


- We can achieve the same effect using a higher-order function:

```
fun {Repeat Count Benchmark}
   Start={OS.time}
in
   for I in 1..Count do {Benchmark} end
   {OS.time}-Start
end
```

- Function Repeat corresponds to method `repeat`
- Procedure argument Benchmark corresponds to method `benchmark`
- With abstract classes, we can achieve the same effect as passing a procedure as argument
  - We use inheritance to simulate a procedure argument

Final classes



- A final class cannot be extended with inheritance

```
final class NotExtendable {
    ...
}
```

- A final method cannot be redefined with inheritance
- It is good practice to define all classes as final classes, except those we wish to be extensible
  - Is it a good idea to define an abstract class as final?