

# Constructors



# The Basics

- You need them!
  - Enables Creation of Objects (new)
- Always write at least one
- Assign your fields from the parameters (or assign default values)

# The Guidelines

- Good Rule of Thumb: Always write a no-argument constructor, even if you don't plan on using it

```
public class A {  
  
    public A() {  
        //code  
    }  
  
}
```

# The Rules

- If you write no constructors for a class, you get the default constructor for free!

```
public class A {
```

```
    //public A() {
```

```
        //code
```

```
    //}
```

```
}
```

# The Rules

- If you write no constructors for a class, you get the default constructor for free!

```
public class A {
```

```
    //public A() {
```

```
        //code
```

```
    //}
```

still has no-arg constructor!

```
}
```

# The Rules

- As soon as you write a constructor, you no longer get the no-arg constructor for free

```
public class Frisbee {  
  
    public Frisbee(int x) {  
        //code  
    }  
  
    //public Frisbee() {  
        //code  
    //}  
  
}
```

# The Rules

- As soon as you write a constructor, you no longer get the no-arg constructor for free

```
public class Frisbee {  
  
    public Frisbee(int x) {  
        //code  
    }  
  
    //public Frisbee() {  
        //code  
    //}  
  
}
```

Frisbee does not have a  
no-argument constructor

# The Rules

- As soon as you write a constructor, you no longer get the no-arg constructor for free

```
public class Frisbee {  
  
    public Frisbee(int x) {  
        //code  
    }  
  
    //public Frisbee() {  
        //code  
    //}  
  
}
```

Frisbee does not have a no-argument constructor

Frisbee ex = new Frisbee();

↑  
compiler error!

# Inheritance Rules

- Constructors are NEVER inherited
- You must always rewrite your constructors (except for the default - which you can potentially get for free)

```
public class Animal {  
  
    public Animal() {  
    }  
  
}
```

```
public class TRex extends Animal {  
  
    public TRex() {  
    }  
  
}
```

# Inheritance Rules

- The first line in every constructor is always a call to a **super** class constructor
- If no such line is written explicitly, then the superclass' no-arg constructor is implicitly called.

```
public class Animal extends Object {  
    public Animal() {  
        super(); //not Object()  
    }  
}  
  
public class Trex extends Animal {  
    public Trex() {  
        //super(); - automatic  
    }  
}
```

# Inheritance Rules

- You can call any super class constructor you want!

```
public class Shape {
```

```
    private int sides;
```

```
    public Shape(int s) {  
        sides = s;  
    }  
}
```

```
    public Shape() {  
        sides = 3;  
    }  
}
```

```
}
```

```
public class Quad extends Shape {
```

```
    public Quad() {  
        super(4);  
        //sets the sides to be 4  
    }  
}
```

```
}
```

# Why super()?

- A subclass cannot directly access the superclass private variables

```
public class Shape {
```

```
    private int sides;
```

```
    public Shape(int s) {  
        sides = s;  
    }  
}
```

```
    public Shape() {  
        sides = 3;  
    }  
}
```

```
}
```

```
public class Quad extends Shape {
```

```
    public Quad() {  
        sides = 4; //ERROR  
    }  
}
```

```
}
```

# Why super()?

- A subclass cannot directly access the superclass private variables

```
public class Shape {
```

```
    private int sides;
```

```
    public Shape(int s) {  
        sides = s;  
    }  
}
```

```
    public Shape() {  
        sides = 3;  
    }  
}
```

```
}
```

```
public class Quad extends Shape {
```

```
    public Quad() {  
        sides = 4; //ERROR  
    }  
}
```

Could use setSides(4) in theory...

# Why super()?

- A subclass cannot directly access the superclass private variables

```
public class Shape {
```

```
    private int sides;
```

```
    public Shape(int s) {  
        sides = s;  
    }  
}
```

```
    public Shape() {  
        sides = 3;  
    }  
}
```

```
}
```

```
public class Quad extends Shape {
```

```
    public Quad() {  
        sides = 4; //ERROR  
    }  
}
```

```
}
```

Could use setSides(4) in theory...

sides would be set to 3 first though, then changed to 4

# Un-override

- What if your class overrides a method and you want to invoke the superclass' old version?

```
public class Name {  
    private String name;
```

```
    public String getName() {  
        return name;  
    }  
}
```

```
public class SuffixedName extends Name {  
    private String suffix;
```

```
    public String getName() {  
        return super.getName() + " " + suffix;  
    }  
}
```

# Un-override

- What if your class overrides a method and you want to invoke the superclass' old version?

```
public class Name {  
    private String name;
```

```
    public String getName() {  
        return name;  
    }  
}
```

```
public class SuffixName extends Name {  
    private String suffix;
```

```
    public String getName() {  
        return super.getName() + " " + suffix;  
    }  
}
```

`super.method();` - calls the superclass' version