A Simple Class

We will use the simple class below for our examples.

```
public class simple
{
  private int num;
  public simple(int a) { num = a; }
  public void set(int a) { num = a;}
  public int get() { return num; }
}
```

- One variable num
- One constructor
- One set method
- One get method

The Setup

Code Starts

```
main();
```

- Calling a method adds a frame to the stack

java heap example

```
</div>
```

Creating a primitive

```
public main()
{
  int i_prim = 3;
}
```

- Primitive variables are stored in a frame

java heap example

```
</div>
```

Creating a object

```


public main()
{
  int i_prim = 3;
  simple s_obj = new simple(5);
}
```

- Objects are stored in the Heap
- A link to the object is stored in the frame
- *A reference*

java heap example

```
</div>
```

Objects in memory

Objects are stored on the heap. When an object is created two memory locations are set up

- In the heap the object is created
- In the current frame the memory address of that object is stored
- `simple q = new simple(9);`
- q stores the memory address, in the heap, of an object of type simple that stores the value 9

Creating an object

```


public main()
{
  int i_prim = 3;
  simple s_obj = new simple(5);
}
```

- Objects are stored in the Heap
- A link to the object is stored in the frame
- *A reference*

java heap example

```
</div>
```

Changing an object

```


public main()
{
  int i_prim = 3;
  simple s_obj = new simple(5);
  s_obj.set(10);
}
```

- Changes to the object are changes to the heap
- The frame is unchanged

java heap example

```
</div>
```

Passing objects as parameters

```


public main()
{
  int i_prim = 3;
  simple s_obj = new simple(5);
  s_obj.set(10);
  test(i_prim, s_obj);
}
public test(int a, simple b)
{
   a = 6;
   b.set(15);
}
```

- Objects can be passed as parameters

java heap example

```
</div>
```

Passing object links as parameters

```
public main()
{
  int i_prim = 3;
  simple s_obj = new simple(5);
  s_obj.set(10);
  test(i_prim, s_obj);
}
public test(int a, simple b)
{
   a = 6;
   b.set(15);
}
```

- Objects can be passed as parameters
- Remember though that it is the reference we are passing
- The link to the heap

java heap example

```
</div>
```

Primitives are passed by value

so only the local value changes

```
public main()
{
  int i_prim = 3;
  simple s_obj = new simple(5);
  s_obj.set(10);
  test(i_prim, s_obj);
}
public test(int a, simple b)
{
   a = 6;
   b.set(15);
}
```

- Changing the primitive parameter changes only the local variable

java heap example

```
</div>
```

Objects are passed by reference

so only the local value changes

```


public main()
{
  int i_prim = 3;
  simple s_obj = new simple(5);
  s_obj.set(10);
  test(i_prim, s_obj);
}
public test(int a, simple b)
{
   a = 6;
   b.set(15);
}
```

- The object changes on the heap so the object changes everywhere

java heap example

```
</div>
```

Objects changes are persistent

They stay changed after the method exits.

```


public main()
{
  int i_prim = 3;
  simple s_obj = new simple(5);
  s_obj.set(10);
  test(i_prim, s_obj);
}
public test(int a, simple b)
{
   a = 6;
   b.set(15);
}
```

-

java heap example

```
</div>
```

Objects as parameters

Objects themselves are not actually passed as parameters. Instead the memory address of the object is passed. The memory address is passed like a primitive variable.

Object reference is a local primitive variable

```


public main()
{
  int i_prim = 3;
  simple s_obj = new simple(5);
  s_obj.set(10);
  test(i_prim, s_obj);
}
public test(int a, simple b)
{
   a = 6;
   b.set(15)
   b = new simple(8);
}
```

-

java heap example

```
</div>
```

Object reference is a local primitive variable

```


public main()
{
  int i_prim = 3;
  simple s_obj = new simple(5);
  s_obj.set(10);
  test(i_prim, s_obj);
}
```

```
public test(int a, simple b)
{
   a = 6;
   b.set(15)
   b = new simple(8);
}
```

- This creates a new object on the heap
- It does not change the original object, nor does it affect the original link

java heap example

```
</div>
```

Object scope

```


public main()
{
  int i_prim = 3;
  simple s_obj = new simple(5);
  s_obj.set(10);
  test(i_prim, s_obj);
}
public test(int a, simple b)
{
   a = 6;
   b.set(15)
   b = new simple(8);
}
```

- When test() exits, the second object is deleted
- Objects are deleted when they have no handles to them

java heap example

```
</div>
```

Garbage collection

```


public main()
{
  int i_prim = 3;
  simple s_obj = new simple(5);
  s_obj.set(10);
  test(i_prim, s_obj);
}
public test(int a, simple b)
{
   a = 6;
   b.set(15)
   b = new simple(8);
}
```

- Technically the object still exists but we have no way to access it
- Java's garbage collection will eventually notice we have no link and delete the object

java heap example

```
</div>
```

Garbage collection

Some Time Later

```
public main()
{
  int i_prim = 3;
  simple s_obj = new simple(5);
  s_obj.set(10);
  test(i_prim, s_obj);
}
public test(int a, simple b)
{
   a = 6;
   b.set(15)
   b = new simple(8);
}
```

- Technically the object still exists but we have no way to access it
- Java's garbage collection will eventually notice we have no link and delete the object

java heap example

```
</div>
```