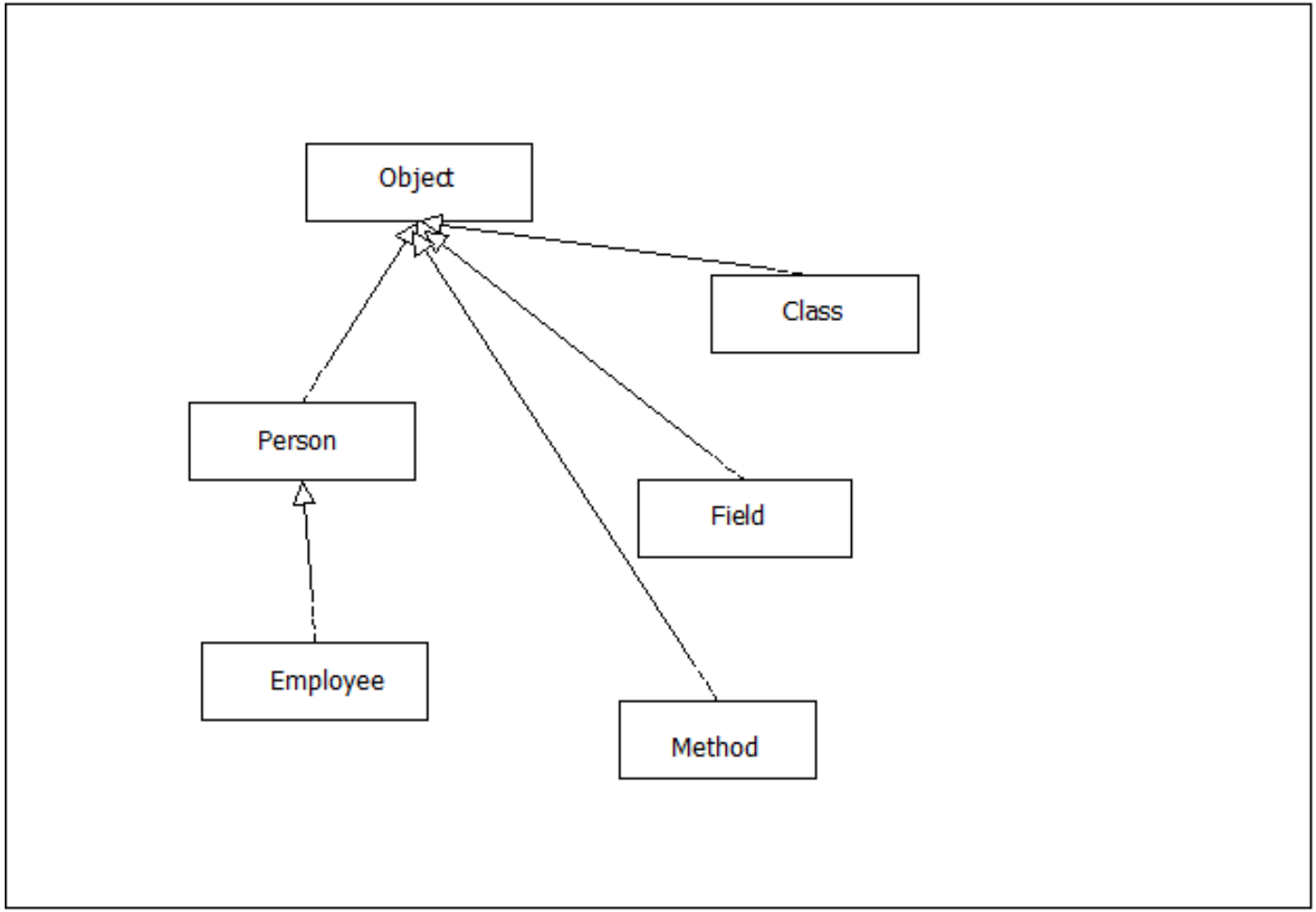
Object – Oriented Technology

Chapter 3: Virtual Platform

Suad Alagić

Springer 2016

REFLECTION



```
public final class Class extends Object {
  public Object newInstance()
        throws InstantiationException, IllegalAccessException;
  public boolean isInstance(Object obj);
  public Class getSuperclass();
  public Class[] getClasses();
  public Field[] getFields();
  public Method[] getMethods();
  public Constructor[] getConstructors();
// other methods
}
```

CLASS METHODS

public Field getField(String name)
    throws NoSuchFieldException
public Method getMethod(String name, Class[] parameterTypes)
    throws NoSuchMethodException
public Constructor getConstructor(Class[] parameterTypes)
    throws NoSuchMethodException.

CLASS METHODS

public String toString()
public String getName()
public int getModifiers()

public boolean isPrimitive()

public boolean isInterface()
public Class[] getInterfaces()

public boolean isAssignableFrom(Class fromClass)

CLASS METHODS

public boolean isArray()
public Class getComponentType()

CLASS METHODS

public static Class forName(String className)
    throws ClassNotFoundException
public ClassLoader getClassLoader()

FIELD OBJECTS

```
public final class Field extends Object {
  public Class getDeclaringClass();
  public String getName();
  public Class getType();
  public Object get(Object obj);
        throws NullPointerException, IllegalArgumentException,
        IllegalAccessException;
  public void set(Object obj, Object value)
        throws NullPointerException, IllegalArgumentException,
        IllegalAccessException;
}
```

METHOD OBJECTS

```
public final class Method extends Object {
  public Class getDeclaringClass();
  public String getName();
  public int getModifiers();
  public Class getReturnType();
  public Class[] getParameterTypes();
  public Class[] getExceptionTypes();
  public Object invoke(Object obj, Object[] args)
        throws NullPointerException, IllegalArgumentException,
        IllegalAccessException, InvocationTargetException;
}
```

CONSTRUCTOR OBJECTS

```
public final class Constructor extends Object {
  public String getName();
  public int getModifiers();
  public Class[] getParameterTypes();
  public Class[] getExceptionTypes();
  public Object newInstance(Object initargs[])
        throws InstantiationException, IllegalArgumentException,
        IllegalAccessException, InvocationTargetException;
}
```

UPDATING FIELDS

```
void updateField(Object o, String fieldName) {
  Object v;
  try {
        Field f = o.getClass().getField(fieldName);
        do {
        v = getObjectOfType(f.getType());
        } while (v = null);
        if (v ≠ null)
        f.set(o, v);
        } catch (Exception e)
        { exception handling }
  }
```

INVOKING METHODS

```
Object invokeMethod(Object o, Method m) {
  Class[] params = m.getParameterTypes();
  Object[] args = new Object[params.length];
  Object newObject = null;
  for(int i = 0; i < params.length; i++) {
  Class cls = params[i];
  args[i] = getObjectOfType(cls);
  try {
        newObject = m.invoke(o, args);
  } catch (Exception e) { exception handling }
  return newObject;
}
```

CREATING CLASS OBJECTS

```
public abstract class ClassLoader {
  protected ClassLoader();
  protected Class loadClass(String Name)
        throws ClassNotFoundException;
  protected Class defineClass(byte[] bytes)
        throws ClassFormatError;
  protected void resolveClass(Class c);
  // other methods
}
```

SPECIAL LOADER

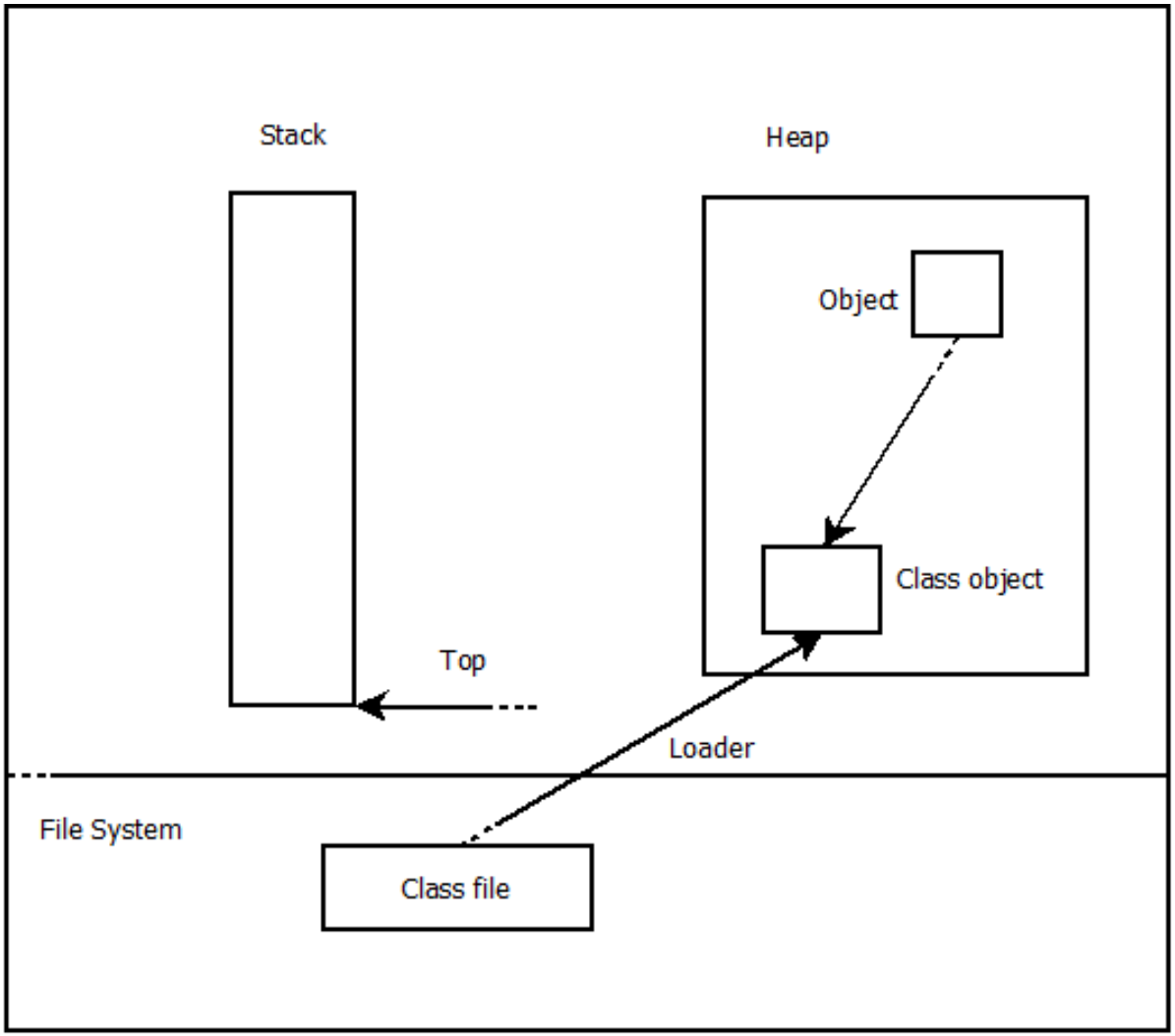
```
class SpecialLoader extends ClassLoader {

  public Class loadClass(String name)
        throws ClassNotFoundException
  {
        try {
        byte[]buffer = getClassBytes(fileName);
        return defineClass(buffer);
        }
        catch (IOException e)
        {exception handling}
// other methods: getClassBytes etc.
}
```

CLASS FILES

```
public class ClassFile {
  // file type and version data
  short          accessFlags;
  CPInfo         constantPool[];
  CPInfo         thisClass;
  CPInfo         superClass;
  CPInfo         interfaces[];
  FieldInfo      fields[];
  MethodInfo     methods[];
  AttributeInfo  attributes[];
// other info
}
```

```
public class MethodInfo {
  short          accessFlags;
  CPInfo         name;
  CPInfo         signature;
  AttributeInfo attributes[];
// other info
}
```

VIRTUAL MACHINE STRUCTURE



VIRTUAL MACHINE STRUCTURE

$$Stack : Object[]$$

$$Top : int$$

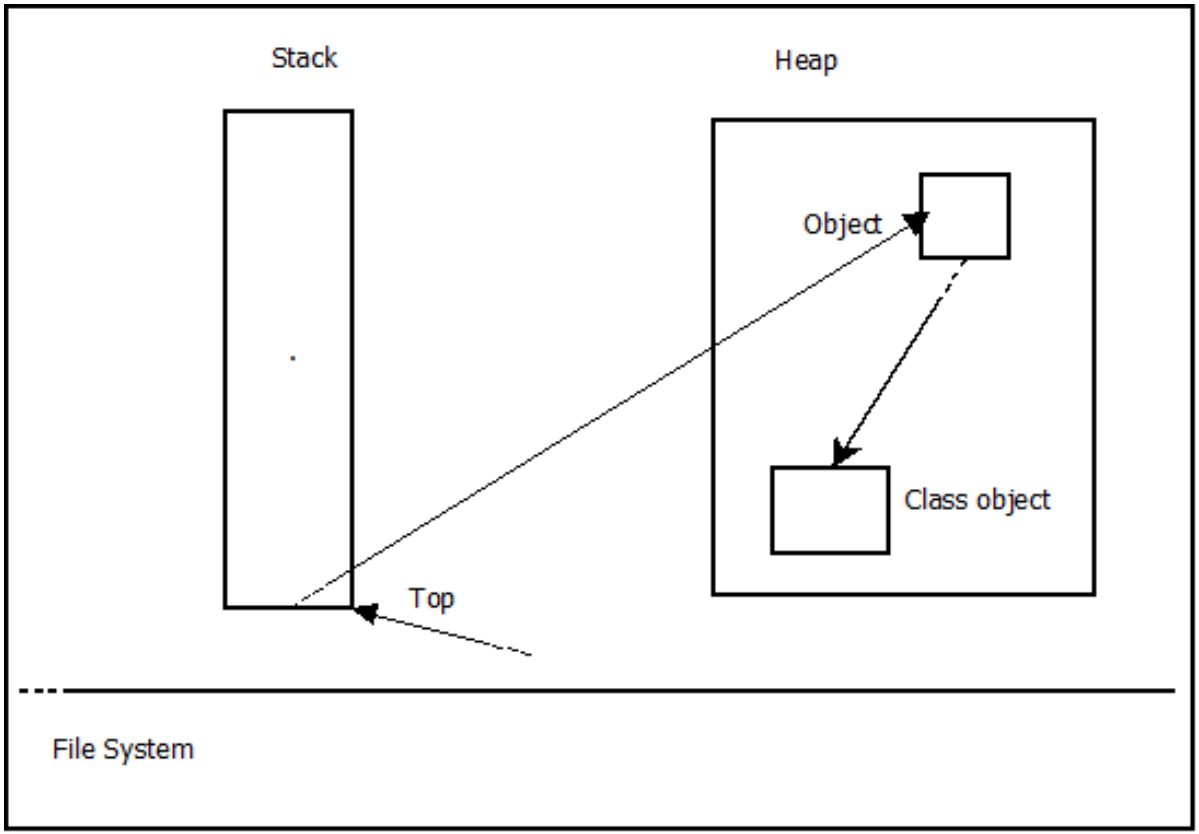
$$Heap : Set < Object >$$

$$classFile : Sequence < byte >$$

$$loader : classFile \rightarrow Class$$

$$VMop : Sequence < byte >$$
$$code : Sequence < VMop >$$

CREATING OBJECTS



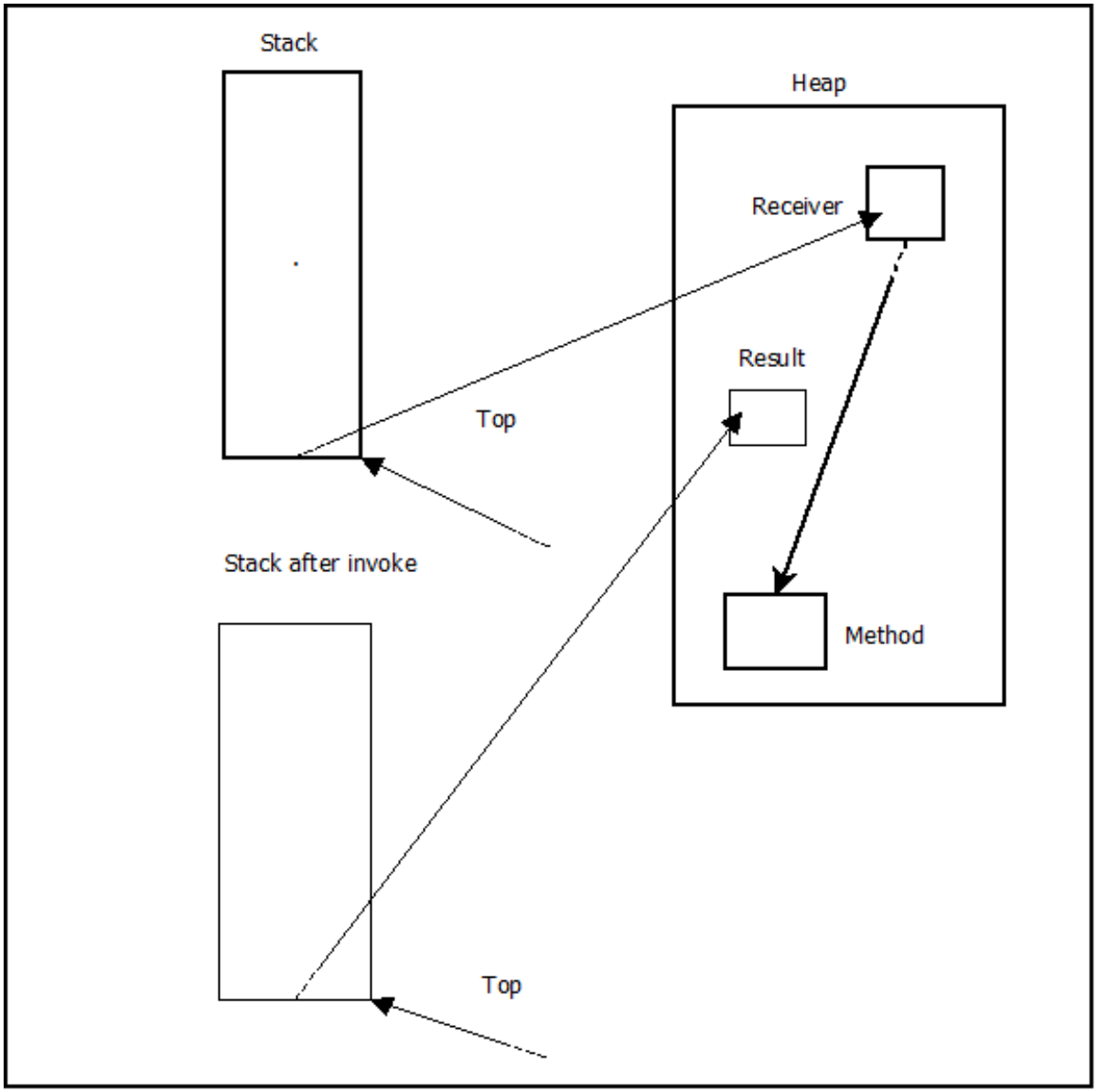
CREATING OBJECTS

$$newInstance : Class \rightarrow Object$$

$$(\exists\, c : Class) \wedge (c \in Heap)$$

$$(\exists\, obj : Object)(obj \in Heap) \wedge$$
$$(newInstance(c) = obj) \wedge (obj.getClass() = c) \wedge$$
$$(Top = \mathbf{old}(Top) + 1) \wedge$$
$$(Stack[Top] = obj)$$

INVOKING METHODS



INVOKING METHODS

$$invokeMethod : Object, Method, Object[] \rightarrow Object$$

$$(\exists\, m : Method)(m \in Heap) \wedge$$
$$(\exists\, r : Object)(Stack[Top] = r) \wedge (r \in Heap) \wedge$$
$$(\exists\, a : Object[])(a \in Heap) \wedge$$
$$(\forall i : int)(i \geq 0) \wedge (i < length(a))(Stack[Top - 1 - i] = a[i])$$

$$(\exists\, obj : Object)(obj \in Heap) \wedge (invokeMethod(r, m, a) = obj) \wedge$$
$$(Top = \textbf{old}(Top) - length(a)) \wedge (Stack[Top] = obj)$$

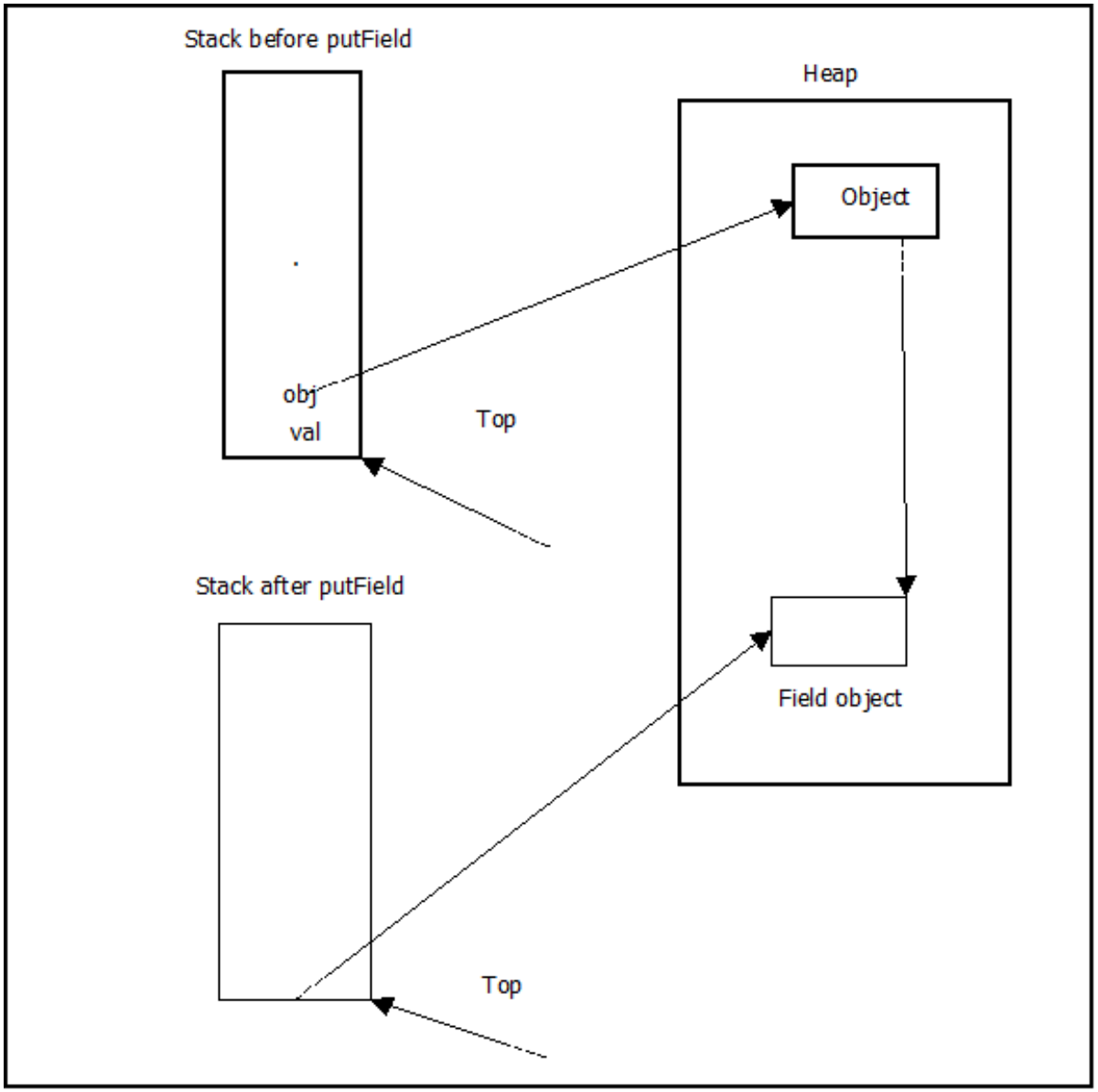
ACCESSING FIELDS

$getField : Field, Object \rightarrow Object$

$(\exists\ f : Field)(f \in Heap) \wedge$
$(\exists\ obj : Object)(obj \in Heap)$

$(\exists\ val : Object)(val \in Heap) \wedge$
$(getField(f, obj) = val) \wedge (Stack[Top] = val) \wedge$
$(\text{Top} = \textbf{old}(\text{Top}) + 1)$

UPDATING FIELDS



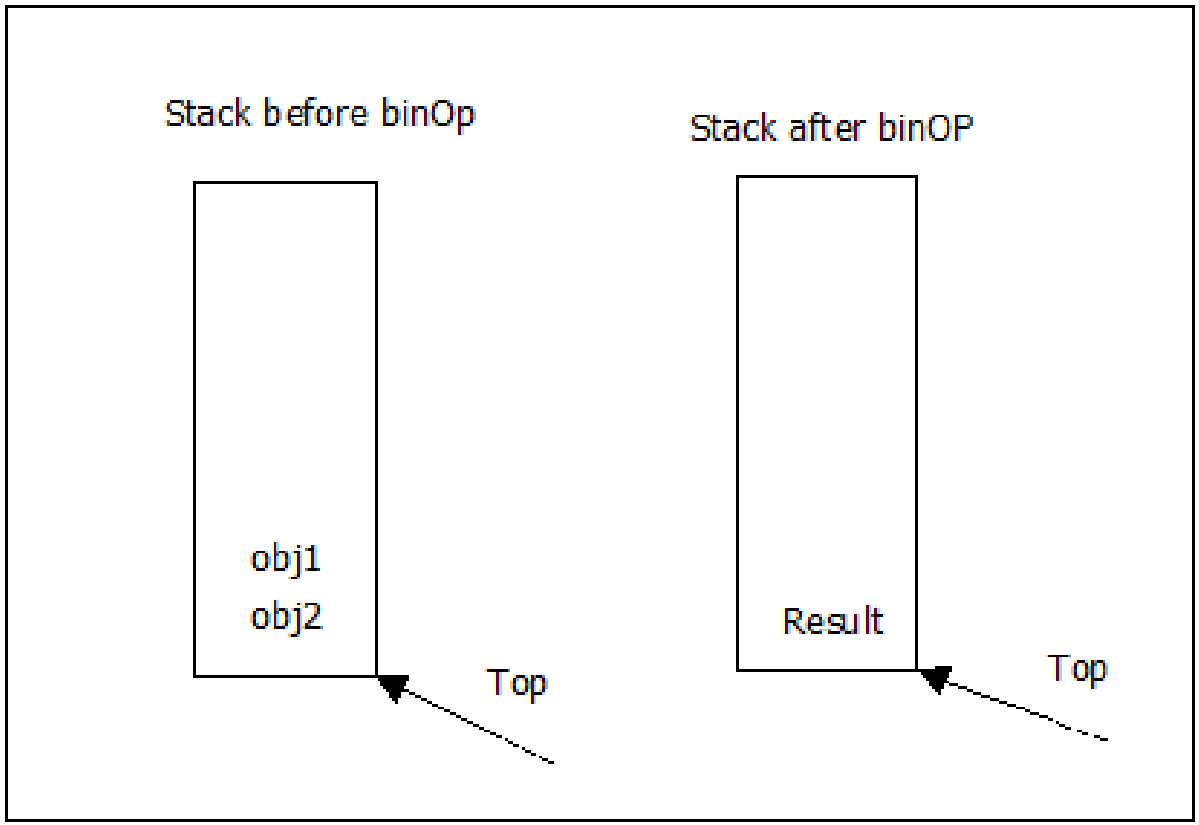
UPDATING FIELDS

$$putField : Field, Object, Object \rightarrow Field$$

$$(\exists\, f : Field)(f \in Heap) \wedge$$
$$(\exists\, obj : Object)(obj \in Heap) \wedge$$
$$(\exists\, val : Object) \wedge$$
$$(Stack[Top] = val) \wedge$$
$$(Stack[Top - 1] = obj)$$

$$(Top = \mathbf{old}(Top) - 1 \wedge$$
$$getField(putField(f, obj, val), obj) = val \wedge$$
$$Stack[Top] = f$$

OPERATIONS


Stack before binOp
obj1
obj2
Top
Stack after binOP
Result
Top

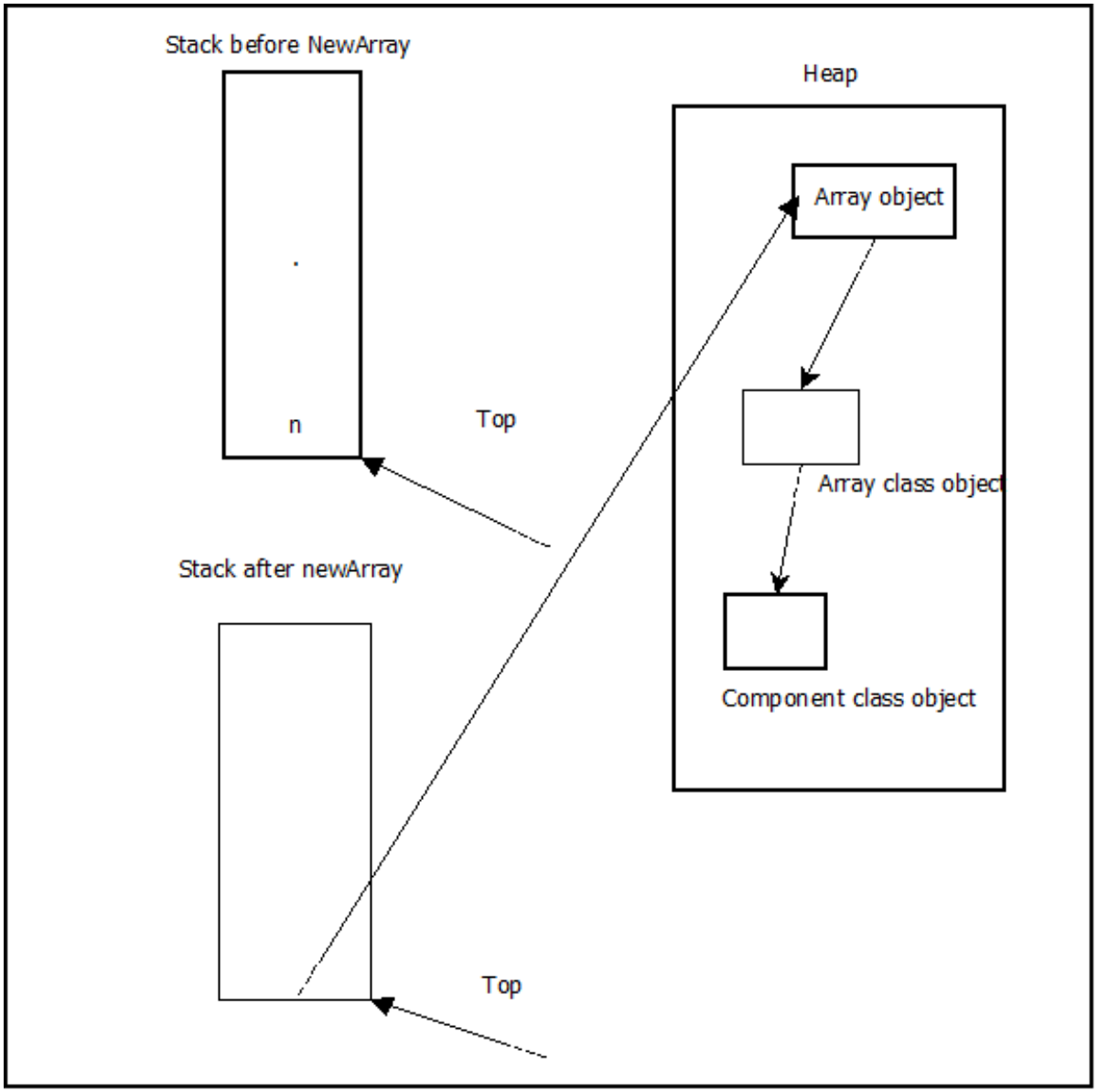
OPERATIONS

$$binaryOp : Object, Object \rightarrow Object$$

$$(\exists\ obj1 : Object)(obj1 \in Heap) \wedge$$
$$(\exists\ obj2 : Object)(obj2 \in Heap) \wedge$$
$$(Stack[Top] = obj1)\ \wedge (Stack[Top - 1] = obj2)$$

$$Top = \textbf{old}(Top) - 1$$
$$Stack[Top] = binaryOp(obj1, obj2)$$

CREATING ARRAYS



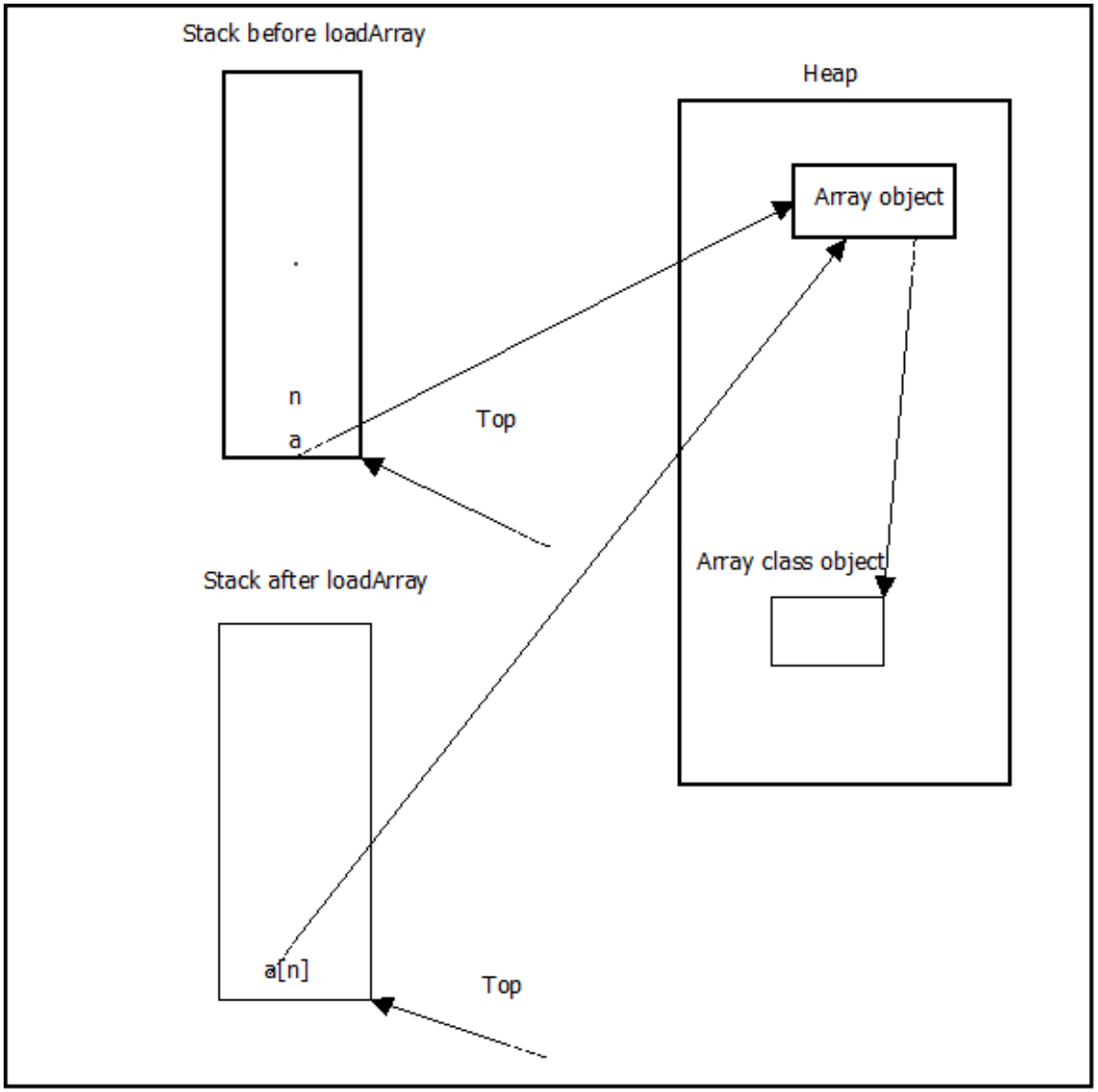
CREATING ARRAYS

$$newArray : Class, Int \rightarrow Object[]$$

$$(\exists\, c : Class)(c \in Heap) \wedge$$
$$(\exists\, n : Int)(Stack[Top] = n)$$

$$(\exists\, obj : Object[])(obj \in Heap) \wedge$$
$$(newArray(c, n) = obj) \wedge (isArray(getClass(obj)) \wedge$$
$$(getComponentType(getClass(obj) = c)) \wedge$$
$$(Stack[Top] = obj) \wedge (length(obj) = n))$$

ACCESSING ARRAYS



ACCESSING ARRAYS

$$arrayLoad : Object[], Int \rightarrow Object$$

$$(\exists\, a : Object[])(a \in Heap) \wedge$$
$$(\exists\, n : Int)(Stack[Top] = a) \wedge (Stack[Top - 1] = n)$$

$$(Top = \textbf{old}(Top) - 1) \wedge$$
$$(\exists\, obj : Object)(arrayLoad(a, n) = obj) \wedge$$
$$(Stack[Top] = obj)$$

UPDATING ARRAYS

$$arrayStore : Object[], Int, Object \rightarrow Object[]$$

$$(\exists\, a : Object[])(a \in Heap) \wedge$$
$$(\exists\, n : Int) \wedge (\exists\, val : Object) \wedge$$
$$(Stack[Top - 2] = a) \wedge (Stack[Top - 1] = n) \wedge$$
$$(Stack[Top] = val)$$

$$(Top = \mathbf{old}(Top) - 3) \wedge$$
$$(arrayLoad(arrayStore(a, n, val), n) = val) \wedge$$
$$(\forall i : Int)(i \neq n)(a[i] = \mathbf{old}(a[i]))$$

ORTHOGONAL PERSISTENCE

- *Orthogonality*
  Persistence is independent of types, i.e., an object (or a value) of any type may be persistent.
- *Transitivity (reachability)*
  If an object is promoted to persistence, so are all of its components, direct or indirect.
- *Transparency*
  The details of the persistent supporting architecture are completely hidden from the users.

```
public interface PJstore {
  public static PJstore getStore();
  public void newProot(String name, Object obj);
  public void setProot(String name, Object obj);
  public Object getProot(String name);
}
```

SAMPLE APPLICATION

```
public class Aircraft
{ private String model;
  private Pilot pilot;
  public Aircraft(String aModel)
  { model = aModel; pilot = null; }
  public assignPilot(Pilot p)
  { pilot=p;}
  // other methods
}
```

SAMPLE APPLICATION

```
public class Pilot
{ private String name;
  private int points;
  public Pilot(String pName, int pPoints)
  { name = pName; points = pPoints; }
  // other methods
}
```

```
public class StoreAircraft {
  Aircraft airObj = new Aircraft("Boeing777");
  Pilot pilot = new Pilot("Mark Sellinger");
  airObj.assignPilot(pilot);
  try {
        PJstore pjs = PJstore.getStore();
        pjs.newProot("MarksPlane", airObj);
  }
  catch (PJexception e) { exception handling }
}
```

UPDATING OBJECTS

```
public class ChangePilot {
  public static void main(String[] args) {
  try {
        PJstore pjs = PJstore.getStore();
        Pilot p= new Pilot("Mark Royer");
        Aircraft airObj = (Aircraft)pjs.getProot("MarksPlane");
        airObj.assignPilot(p);
        }
        catch (PJexception) { exception handling }
  }
}
```

ACCESSING OBJECTS

```
public class DisplayAircraft {
  public static main(String[] args) {
  try {
        PJstore pjs = PJstore.getStore();
        Aircraft airObj = (Aircraft) pjs.getProot("MarksPlane");
        airObj.display();
        }
        catch (PJexception e) { exception handling }
  }
}
```

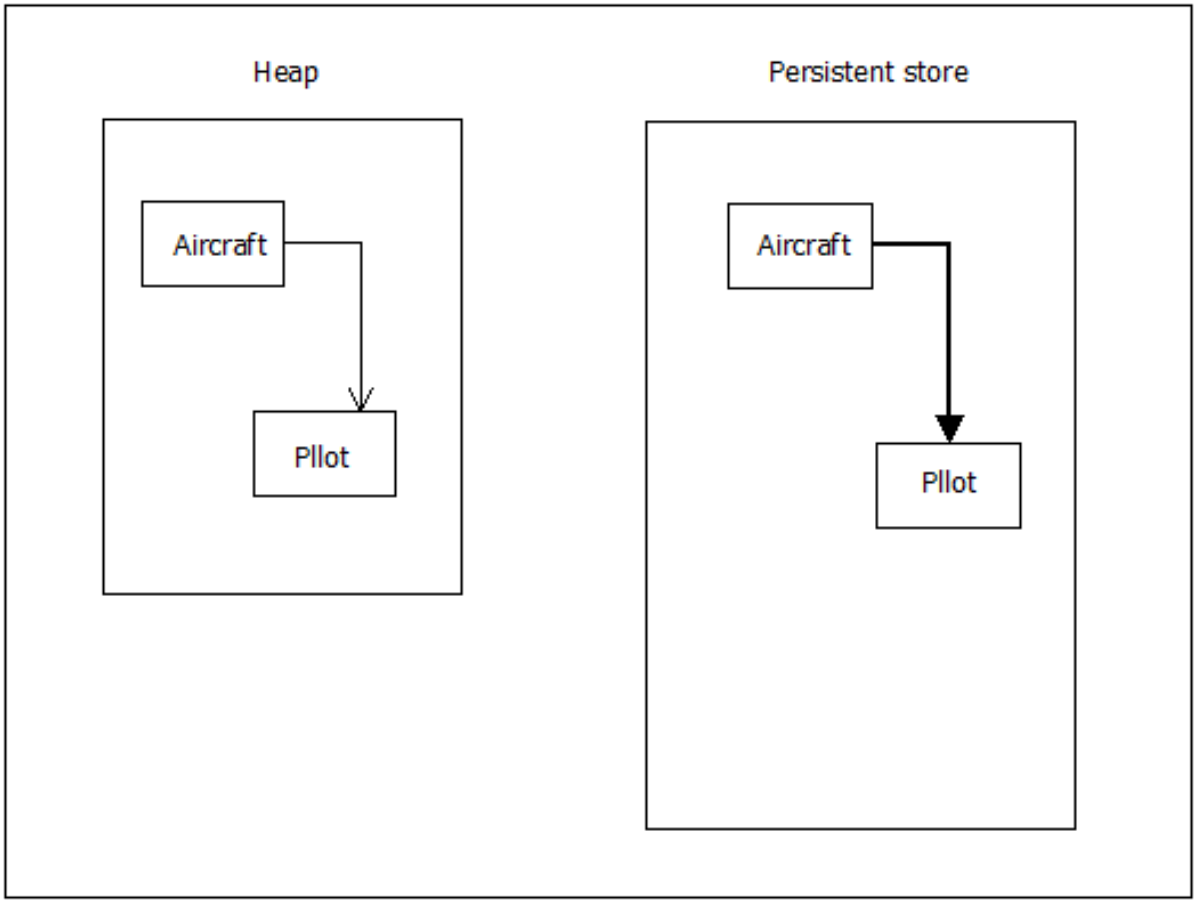
OBJECT METHODS

public boolean existProot(String name)
public Enumeration getAllProotNames()
public Enumeration getAllProots()
public Object getProot(String name)
public void newProot(String name, Object obj)
public void discardProot(String name)

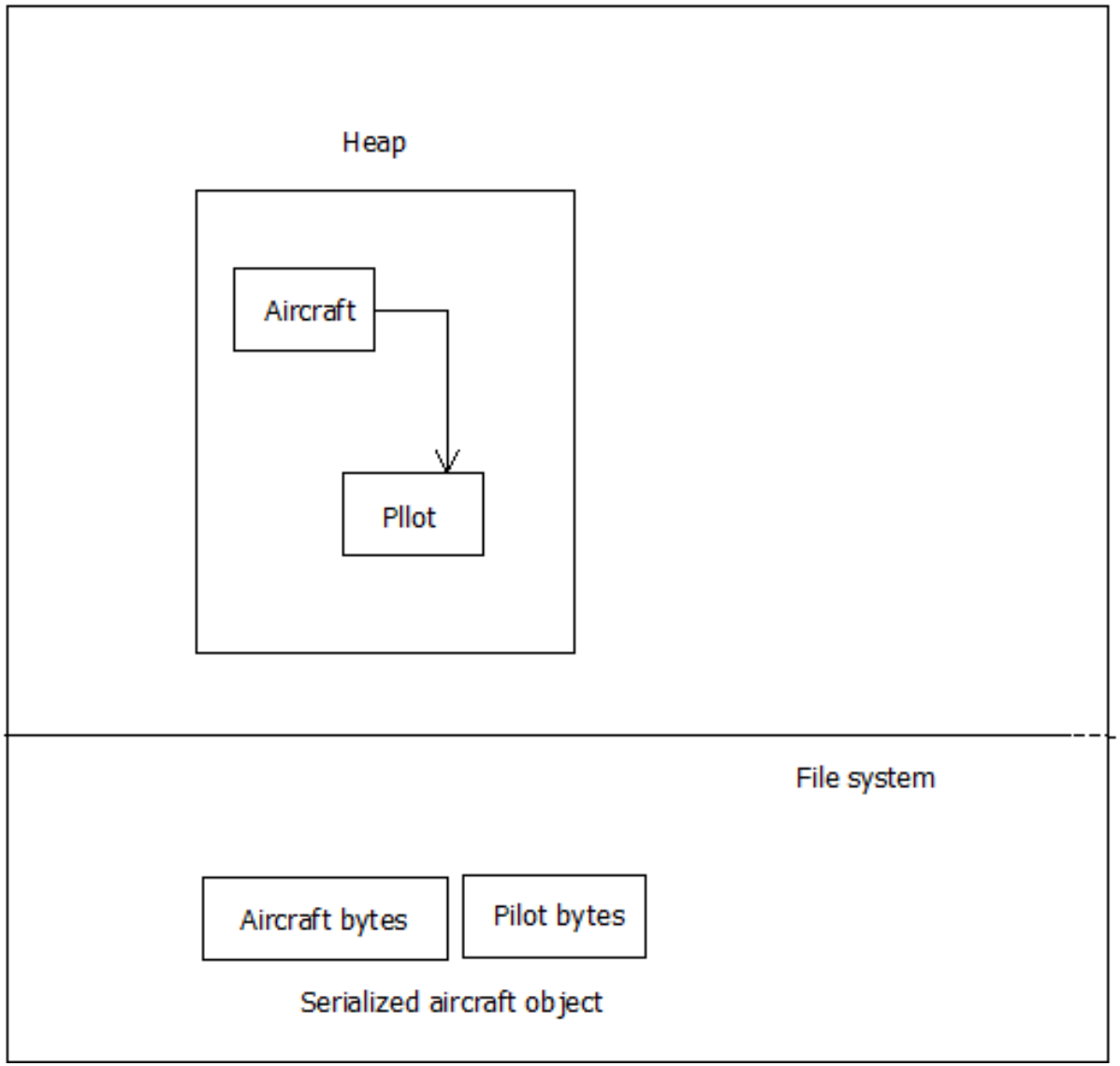
CLASS METHODS

public Enumeration getAllPclassNames()
public Enumeration getAllPclasses()
public boolean existsPclass(String name)
public Class getPclass(String name)

PERSISTENT ARCHITECTURE



SERIALIZATION



OBJECT OUTPUT

```
public interface ObjectOutput extends DataOutput
{
  void writeObject(Object obj);
          throws IOexception;
  // other methods
}
```

OBJECT INPUT

```
public interface ObjectInput extends DataInput
{
  Object readObject()
        throws ClassNotFoundException, IOexception;
  // other methods
  }
```

OBJECT OUTPUT

```
public class Aircraft extends Serializable {
// . . .
}
```

```
FileOutputStream fileOut = new FileOutputStream("AircraftFile");
ObjectOutput out = new ObjectOutputStream(fileOut);
aircraftObj = new Aircraft("Boeing777"); out.writeObject(aircraftObj);
out.flush();
out.close();
```

OBJECT INPUT

```
FileInputStream fileIn = new FileInputStream("AircraftFile");
ObjectInput in = new ObjectInputStream(fileIn);
Aircraft aircraftObj= (Aircraft) in.readObject();
in.close();
```