NETFLIX

Netflix's SimianArmy

Open Source Project
Group 5

amazon web services™

Outline

- Introduction / Overview of Simian Army (George)
  * including System Architecture
- Initialization (David)
- Chaos Monkey (Allen)
- Conformity Monkey (Amity)
- Janitor Monkey (David)
- Client Module and multi-thread handling (Terry)
- Crawler Module (Joe)
- Utilities Module (George, David)
- Coding Style (Whole Group)

Background Information

- Created by Netflix after moving to AWS ***to improve availability & reliability***.
- A cloud architecture, where ***individual components fail but do not affect the availability of the entire system***, was greatly needed.
- Infrastructure is never 100% reliable but stream*ing* must be non-stop.
- SimianArmy deploys “monkeys” to ***make cloud service less fragile and better able to support continuous service*** when some parts have issues.
- Potential weaknesses and/or problems could be detected and addressed.

Analogy by Netflix

- Is your spare tire properly inflated when necessary?
- How do you know? Do you have the tools to change it? Can you?
- One way to guarantee the above questions is to poke a hole in your tire once a week and go through the drill of replacing it.
- Expensive and time-consuming in the real world.
- Almost **free and automated in the cloud.**
- This philosophy led to development of SimianArmy.

*Source: < http://techblog.netflix.com/2011/07/netflix-simian-army.html >

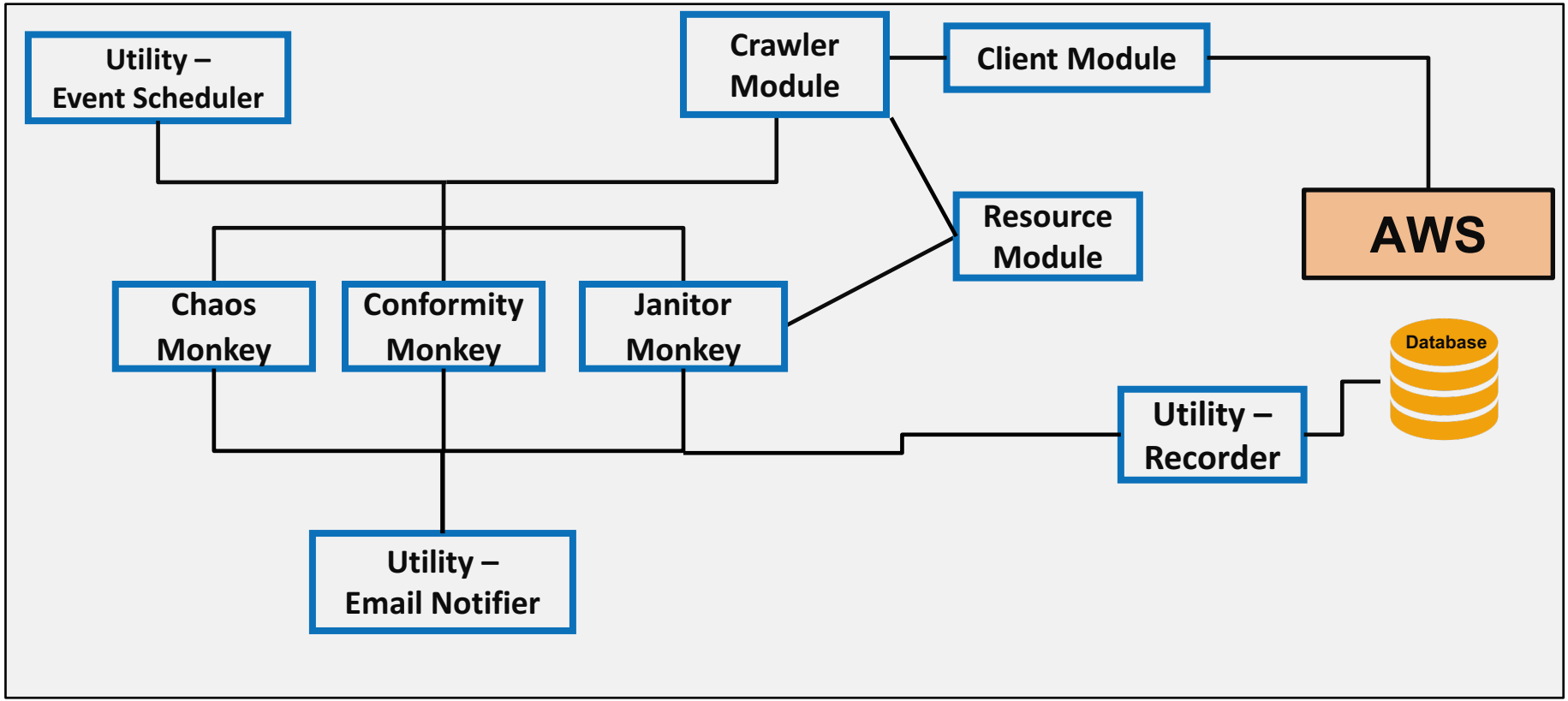
Members of the SimianArmy

- **Chaos Monkey** – randomly shuts down VMs to ensure that small disruptions will not affect the overall service.
- **Conformity Monkey** – detects instances that aren't coded to best-practices and shuts them down, giving the service owner the opportunity to re-launch them properly.
- **Janitor Monkey** – searches for unused resources and discards them.
- *Security Monkey* – searches security weaknesses, and ends the offending instances.
- *Doctor Monkey* – performs health checks on each instance and monitors other external signs of process health such as CPU and memory usage.
- *Latency Monkey* – simulates a degradation of service and checks to make sure that upstream services react appropriately.

Architecture Expression

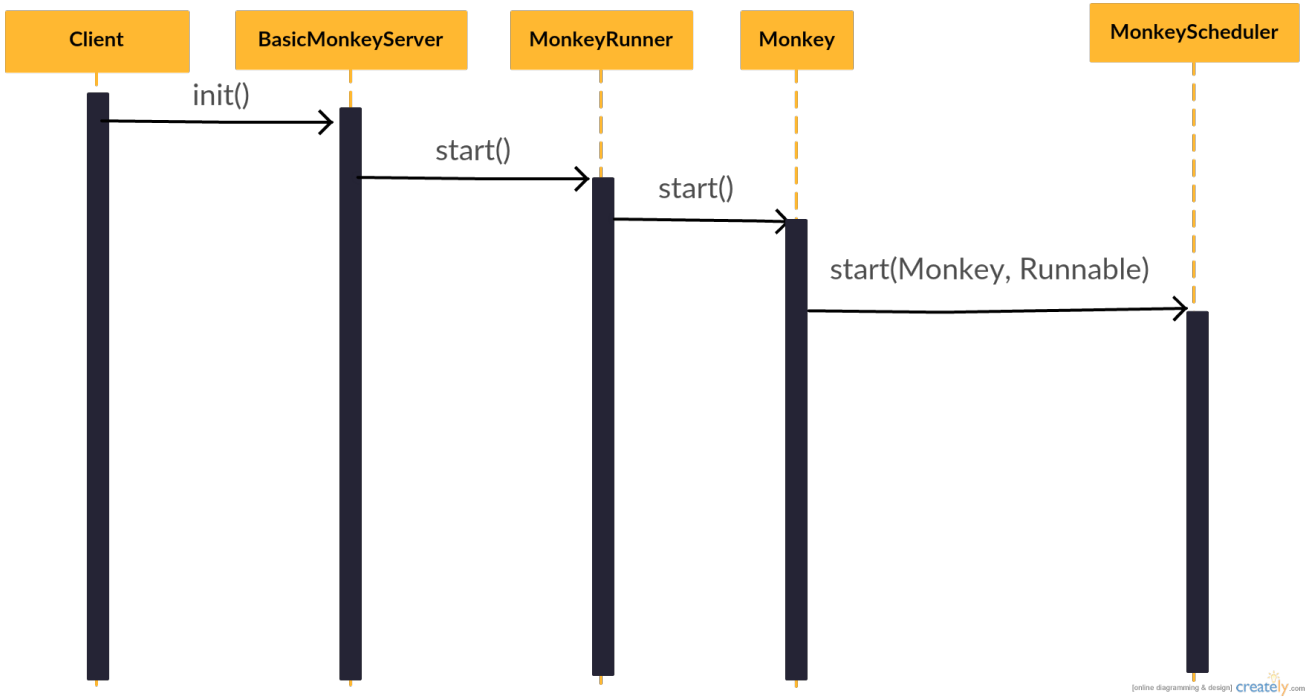
Module	Descriptions
Client Module	Provides functionality/interface that allows the monkeys to interact with the cloud.
Resource Module	Provides functionality of getting the common properties of a resource and the methods to add and retrieve additional properties of one.
Chaos Monkey Module	Provides functionality of creating different types of chaos on AWS, thus causing small disruptions and testing ability to withstand such unexpected events.
Janitor Monkey Module	Provides functionalities of searching for unused resources, marking of them, and cleaning up.
Conformity Monkey Module	Provides functionalities of getting clusters, as well as applying a set of rules to perform conformity checks. A cluster is the basic unit of conformity check.
Crawler Module	Provides functionality of getting auto-scaling-groups from AWS. Retrieves information of such instances, for usages by Janitor Monkey.
Utilities Module	Provides functionalities of scheduling monkeys deployment, recording and finding events in data storage as well as notifying admins via email.

System Architecture Diagram

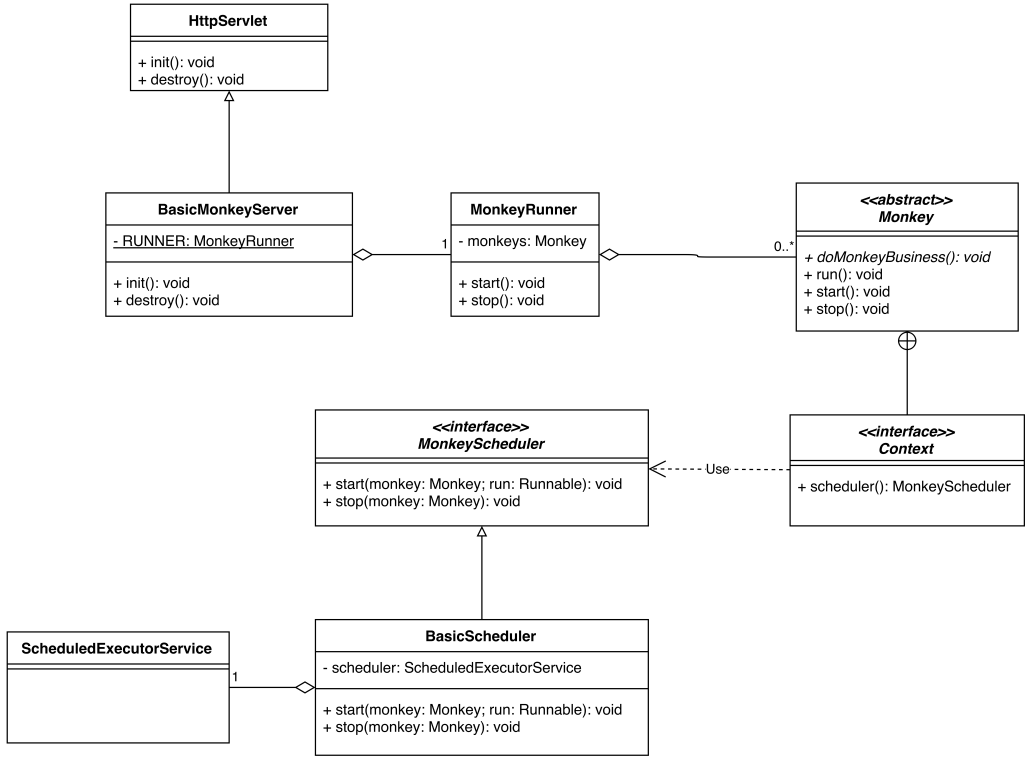


Initialization

Sequence Diagram



Class Diagram

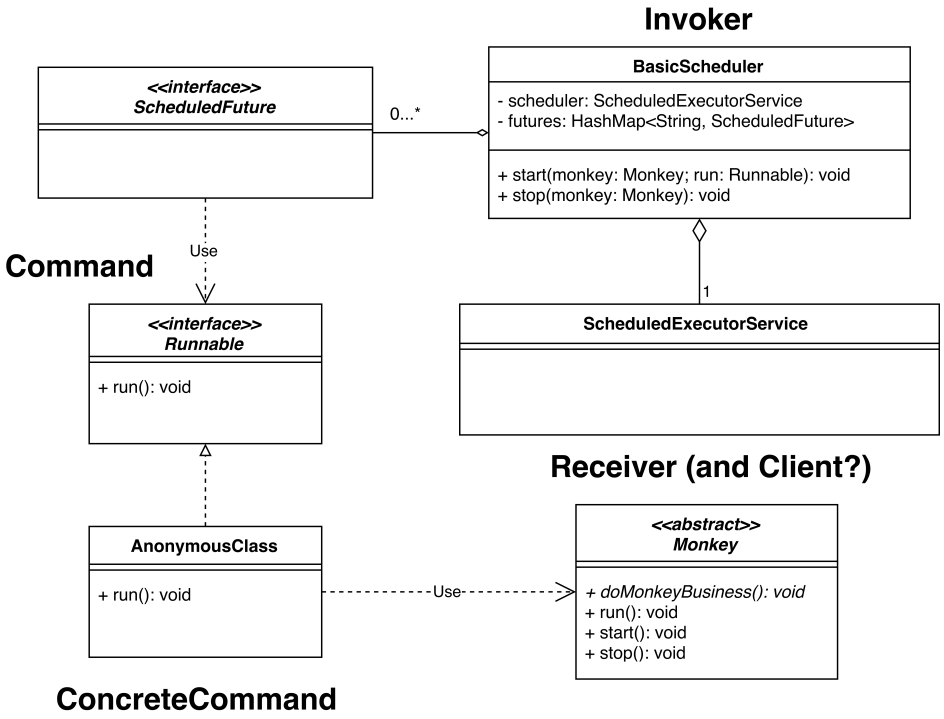


Command Pattern?

- Receiver (Monkey) with an action (run) ?

```
public void start() {
    final Monkey me = this;
    ctx.scheduler().start(this, new Runnable() {
        @Override
        public void run() {
            try {
                me.run();
            } catch (Exception e) {
                LOGGER.error(me.type().name() + " Monkey Error: ", e);
            }
        }
    });
}
```

Command Pattern – Class Diagram



Singleton Class

- MonkeyRunner

```
public enum MonkeyRunner {

    /** The instance. */
    INSTANCE;

    /** The Constant LOGGER. */
    private static final Logger LOGGER = LoggerFactory.getLogger(MonkeyRunner.class);

    /**
     * Gets the single instance of MonkeyRunner.
     *
     * @return single instance of MonkeyRunner
     */
    public static MonkeyRunner getInstance() {
        return INSTANCE;
    }
```

Client

- BasicMonkeyServer

```
public class BasicMonkeyServer extends HttpServlet {
    private static final Logger LOGGER = LoggerFactory.getLogger(BasicMonkeyServer.class);

    private static final MonkeyRunner RUNNER = MonkeyRunner.getInstance();
```

```
@Override
public void init() throws ServletException {
    super.init();
    configureClient();
    addMonkeysToRun();
    RUNNER.start();
}
```

Role of MonkeyRunner

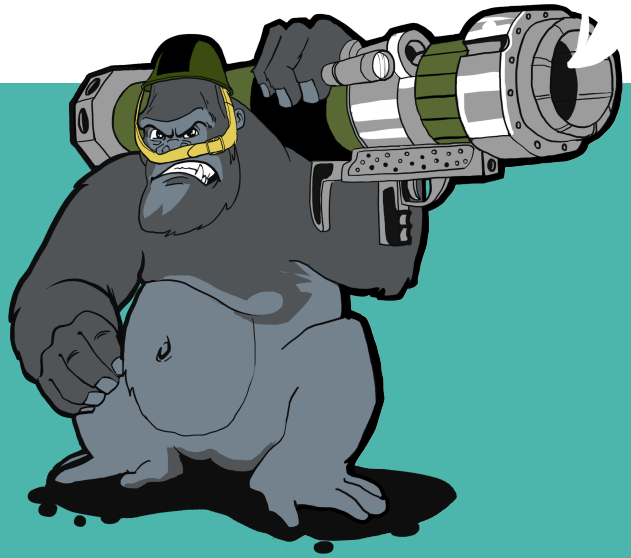
- In charge of all monkeys

```
/**
 * Start all the monkeys registered with addMonkey or replaceMonkey.
 */
public void start() {
    for (Monkey monkey : monkeys) {
        LOGGER.info("Starting " + monkey.type().name() + " Monkey");
        monkey.start();
    }
}

/**
 * Stop all of the registered monkeys.
 */
public void stop() {
    for (Monkey monkey : monkeys) {
        LOGGER.info("Stopping " + monkey.type().name() + " Monkey");
        monkey.stop();
    }
}
```

Enum Singleton

- Easy to write.
- Lazy loaded.
- Thread-safe.
- Automatically handle serialization.

Chaos Monkey



Outline

- What is Chaos Monkey
  - Identifies groups of systems
  - Randomly terminates one of the systems in a group
- Why is Chaos Monkey required
  - Failures happen
  - Even if your architecture can tolerate a system failure,
    are you sure it will still be able to next week, next month?
  - Do your traffic load balancers correctly detect and route requests around system failures?

Design

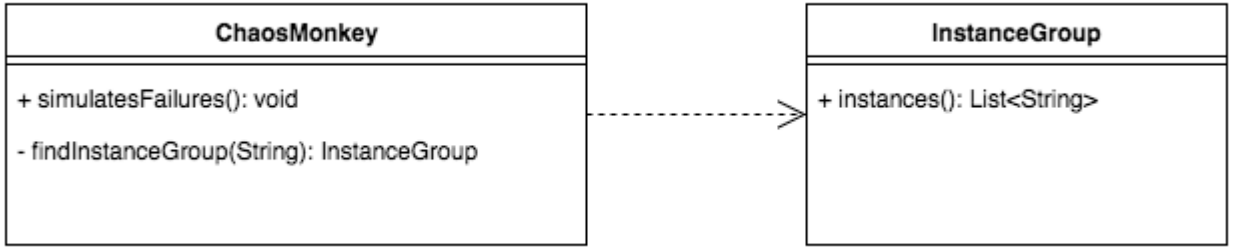
- Design Pattern:
  - Strategy Pattern
- Design Principles:
  - Encapsulate what varies.
  - Depend on abstractions. Do not depend on concrete classes.

Requirements Statements

- Chaos Monkey is a tool which simulates failures on a group of AWS instances randomly.
- Chaos Monkey can generate failures with different types of chaos, such as **Burn-CPU-Chaos** which can make CPU always busy, **Fill-Disk-Chaos** which can make disk out of space, **Block-Network-Chaos** which can make the network disable.
- Chaos Monkey should check that the chaos can be apply on instance before generate failures.
- Some of Chaos Type are implemented with running a script over SSH, some of Chaos Type are implemented with calling AWS's API.

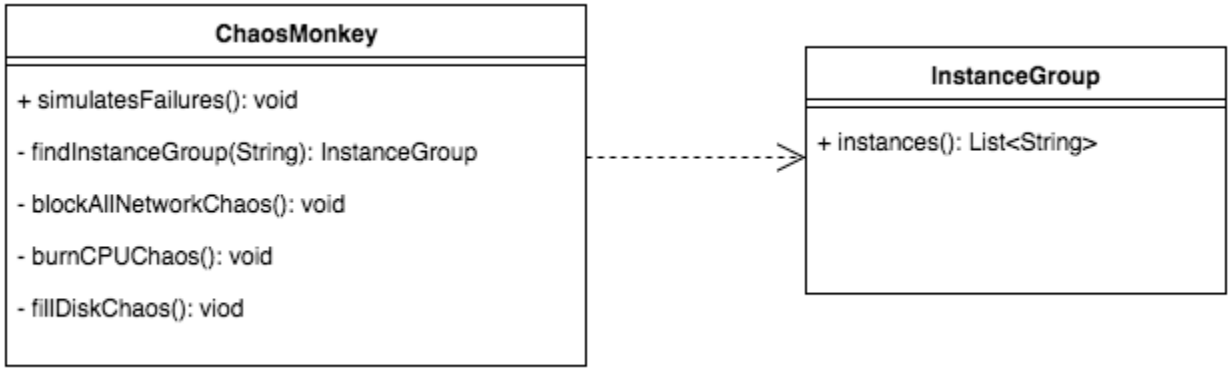
Requirements Statements$_1$

- Chaos Monkey is a tool which simulates failures on a group of AWS instances randomly.



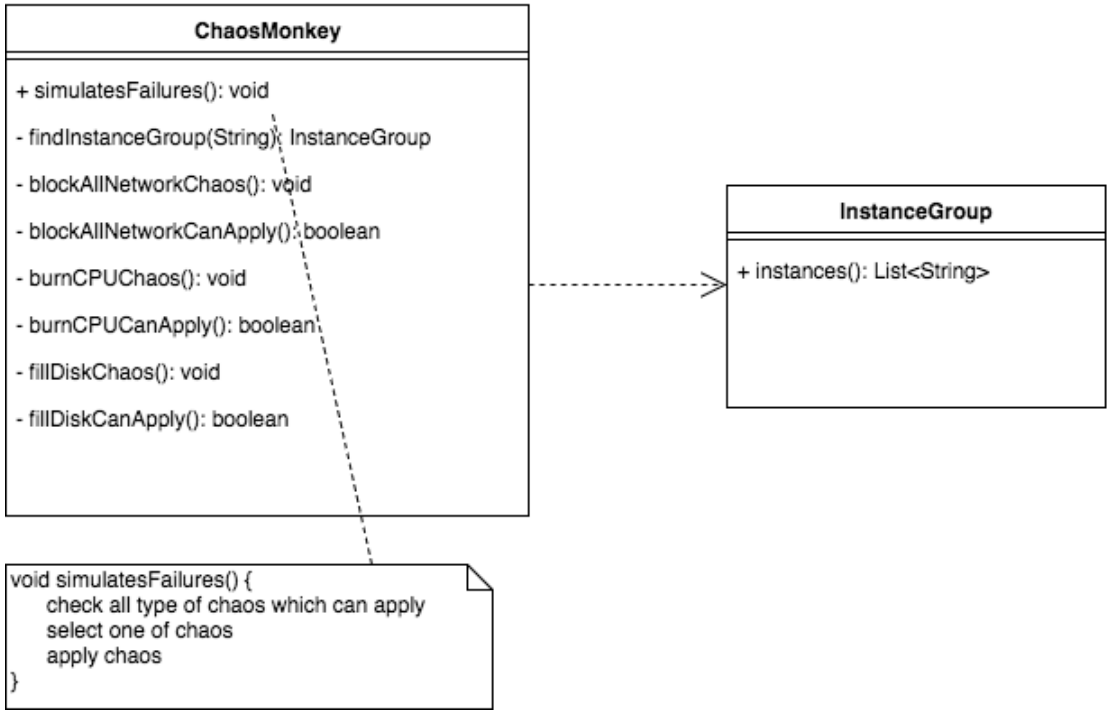
Requirements Statements$_2$

- Chaos Monkey can generate failures with different types of chaos, such as **Burn-CPU-Chaos** which can make CPU always busy, **Fill-Disk-Chaos** which can make disk run out of space, **Block-Network-Chaos** which can make the network disable.



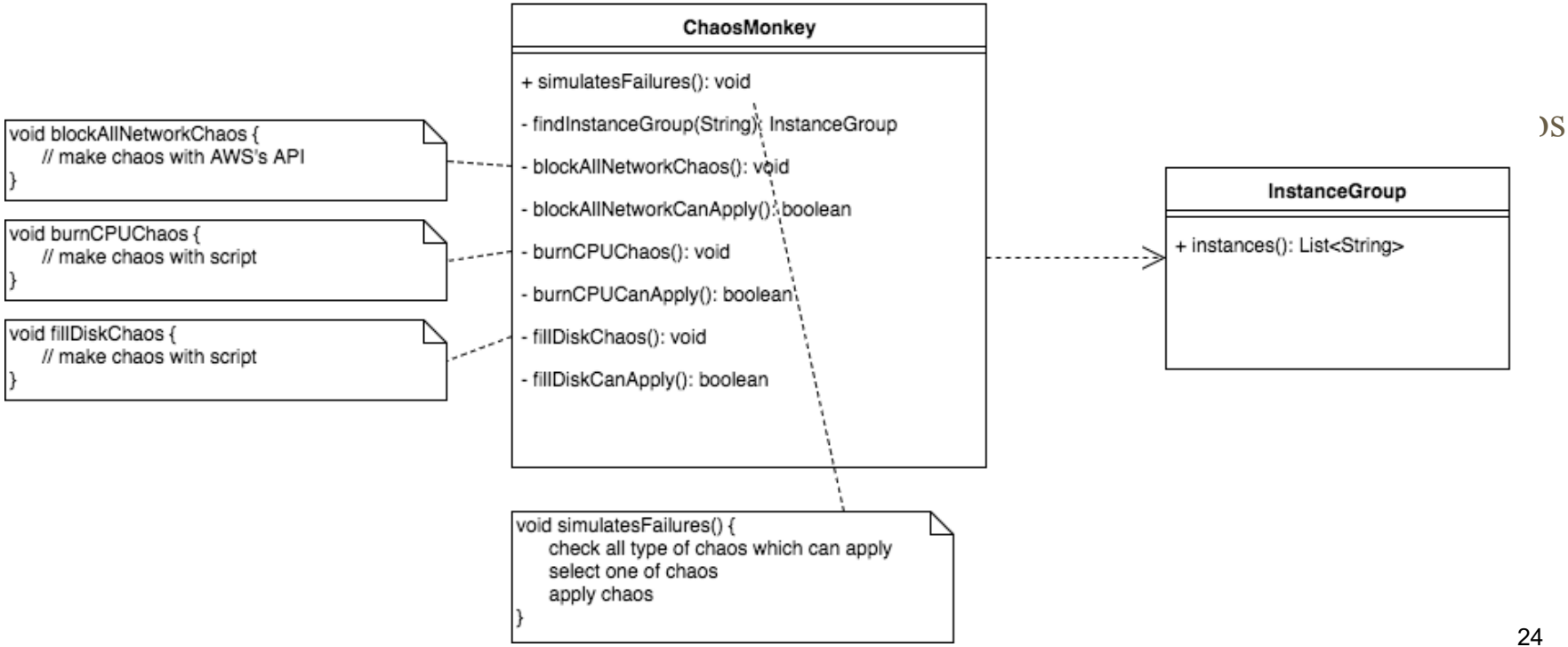
Requirements Statements$_3$

- Chaos Monkey should check that the chaos can be applied on instance before generating failures.

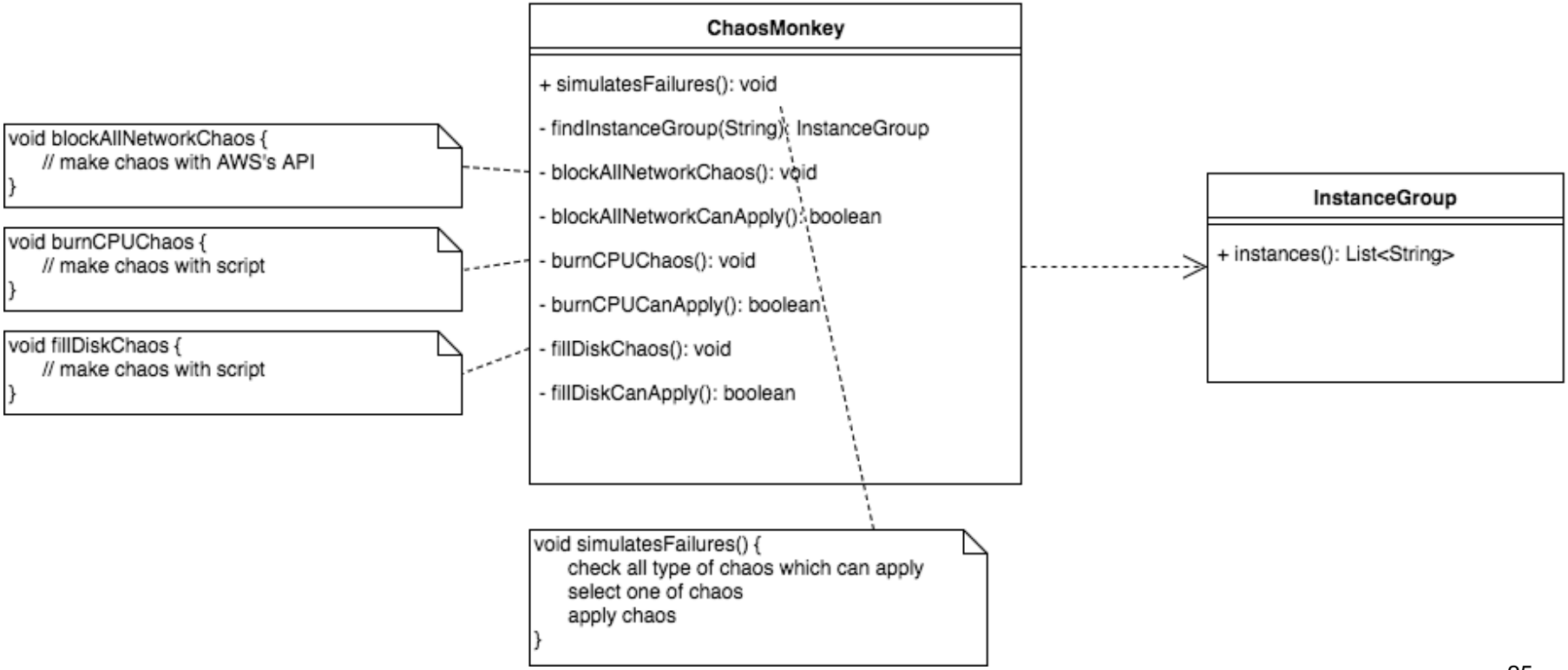


Requirements Statements$_4$

- Some of Chaos Type are implemented with running a script over SSH, some of Chaos Type are implemented with calling AWS's API.

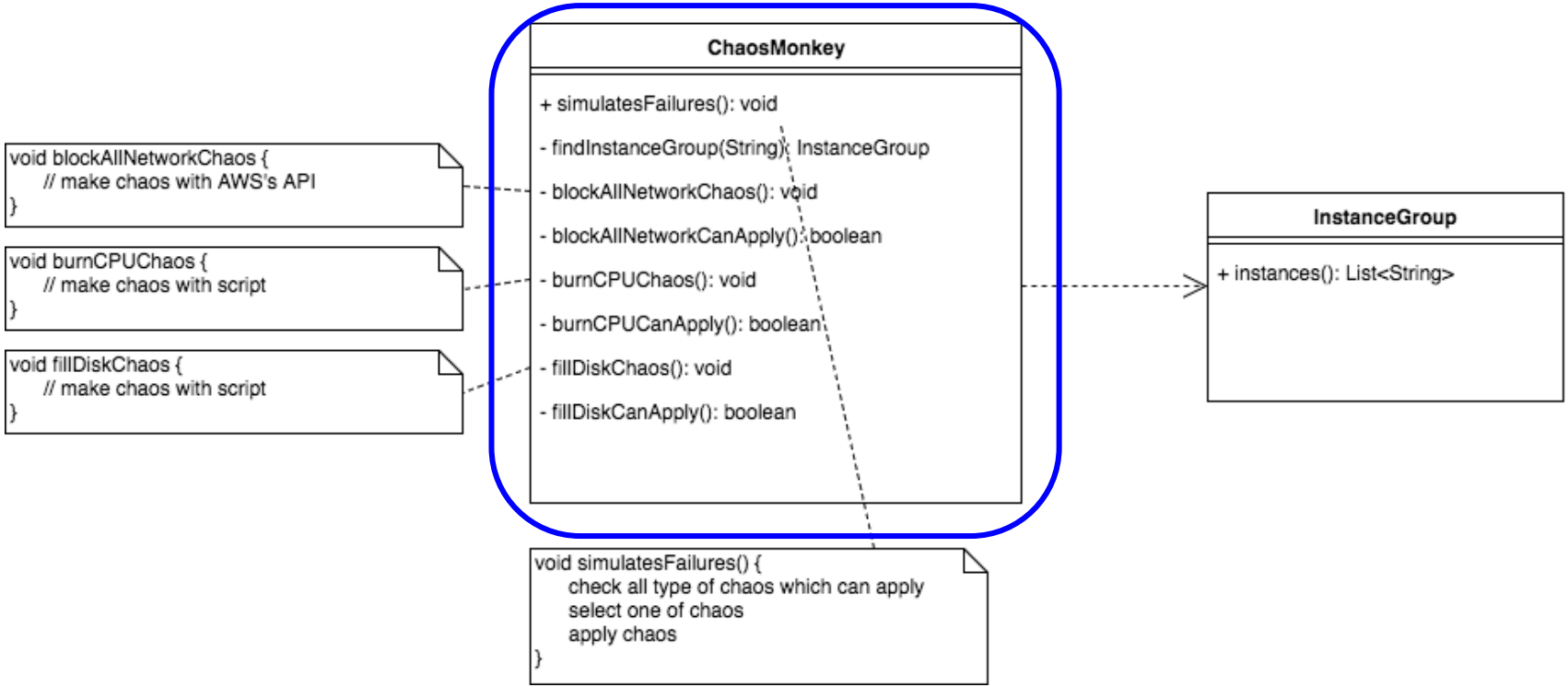


Initial Design

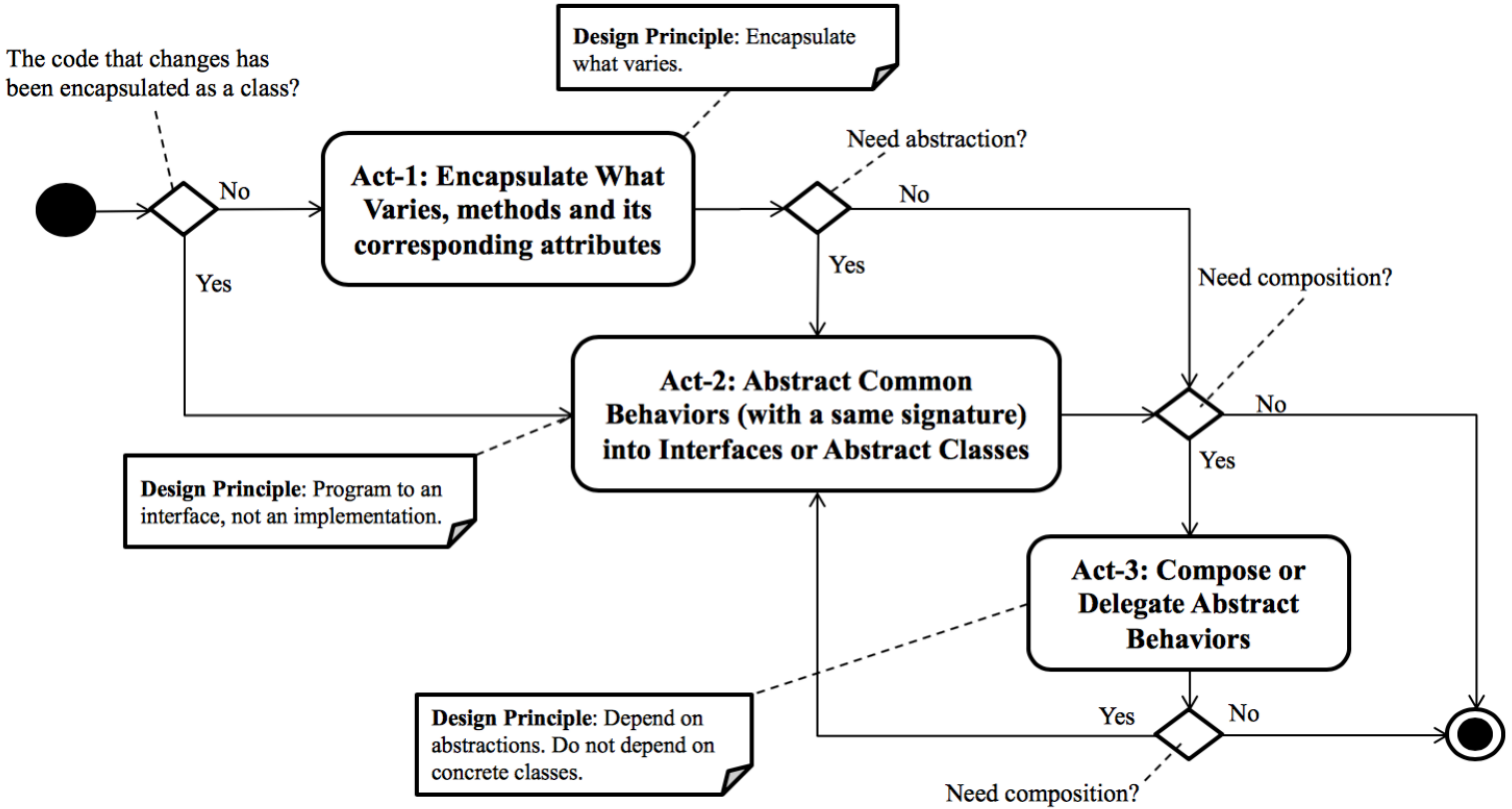


Problem with Initial Design

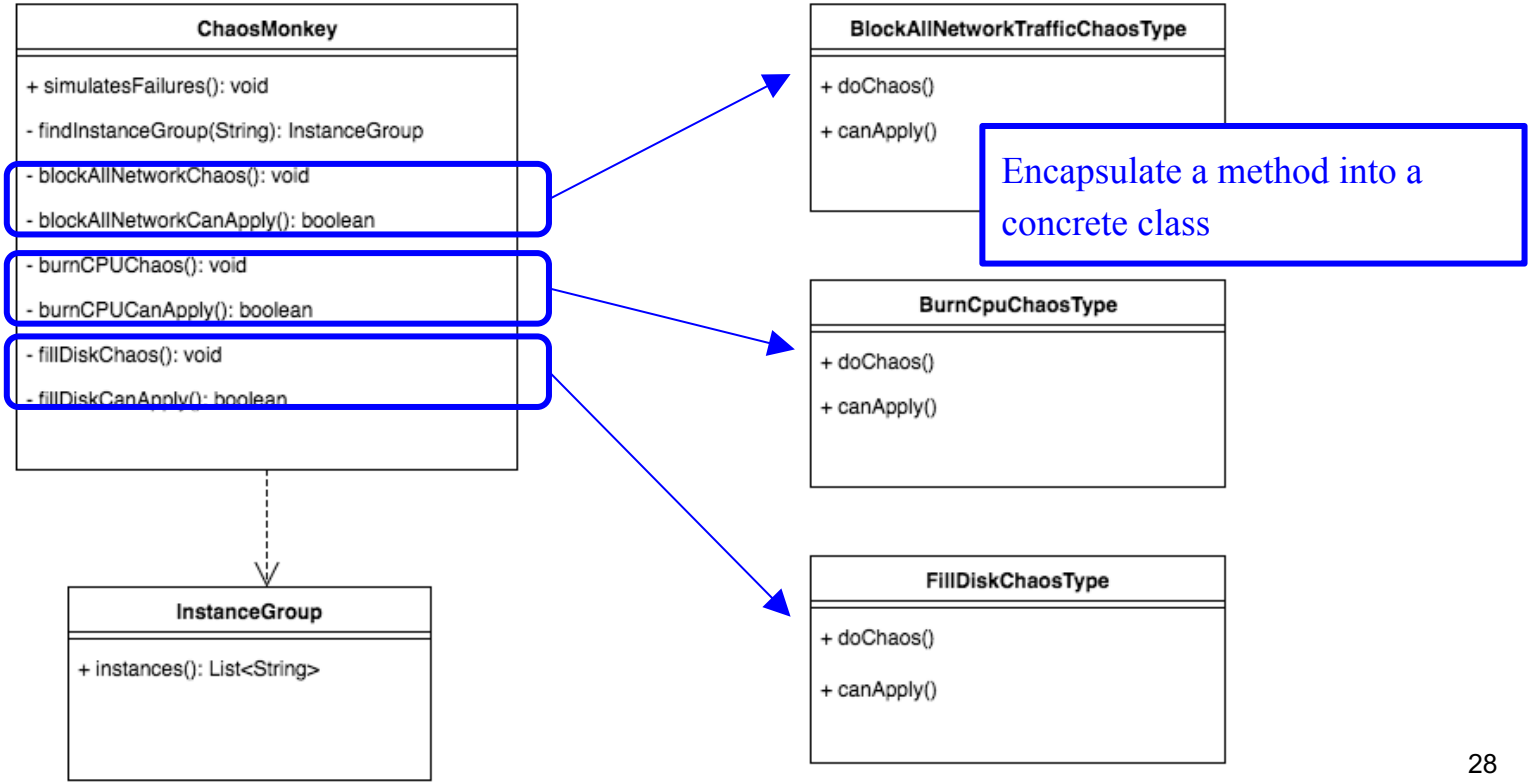
Problem: It's difficult to add new type of Choas and vary existing ones when Chaos is an integral part of ChaosMonkey.



Design Process for Change

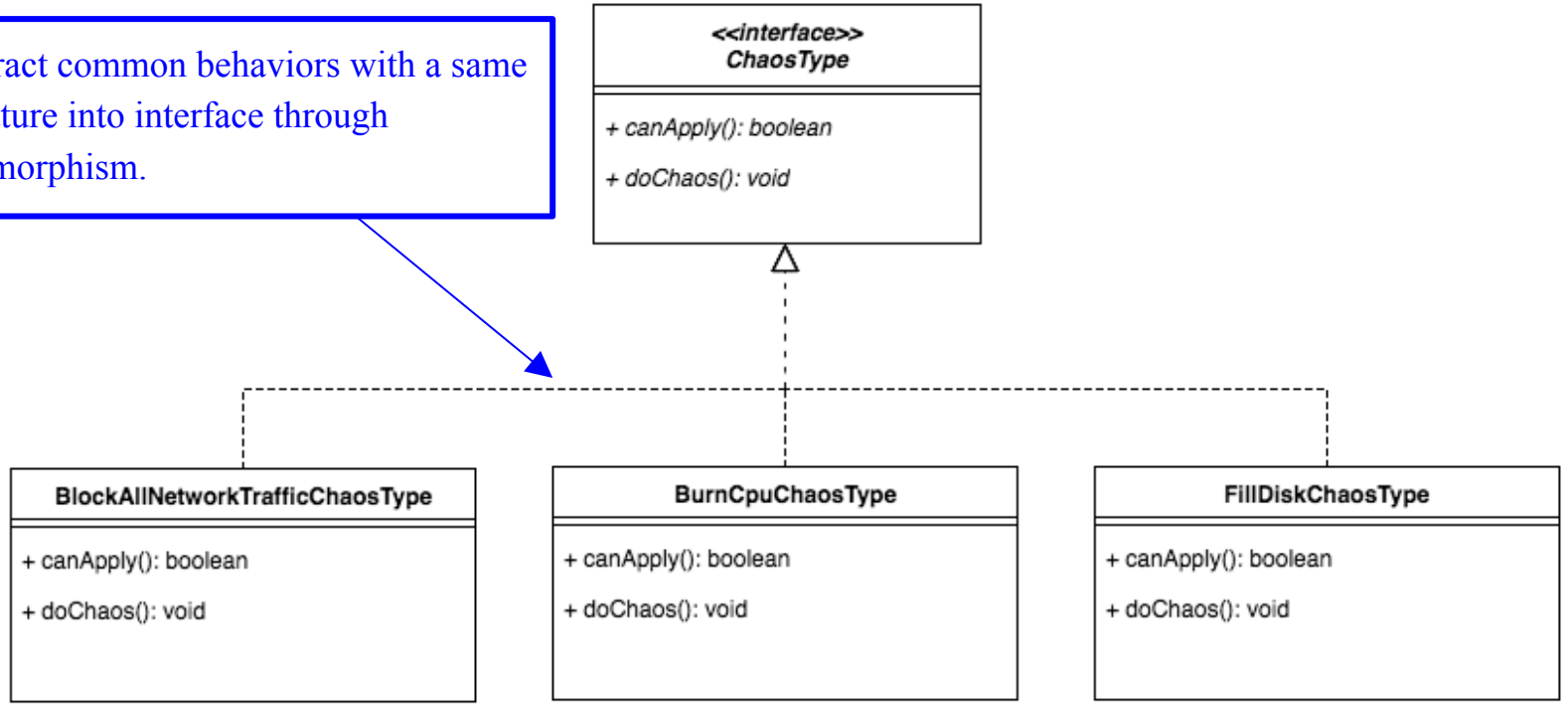


Design Process for Change

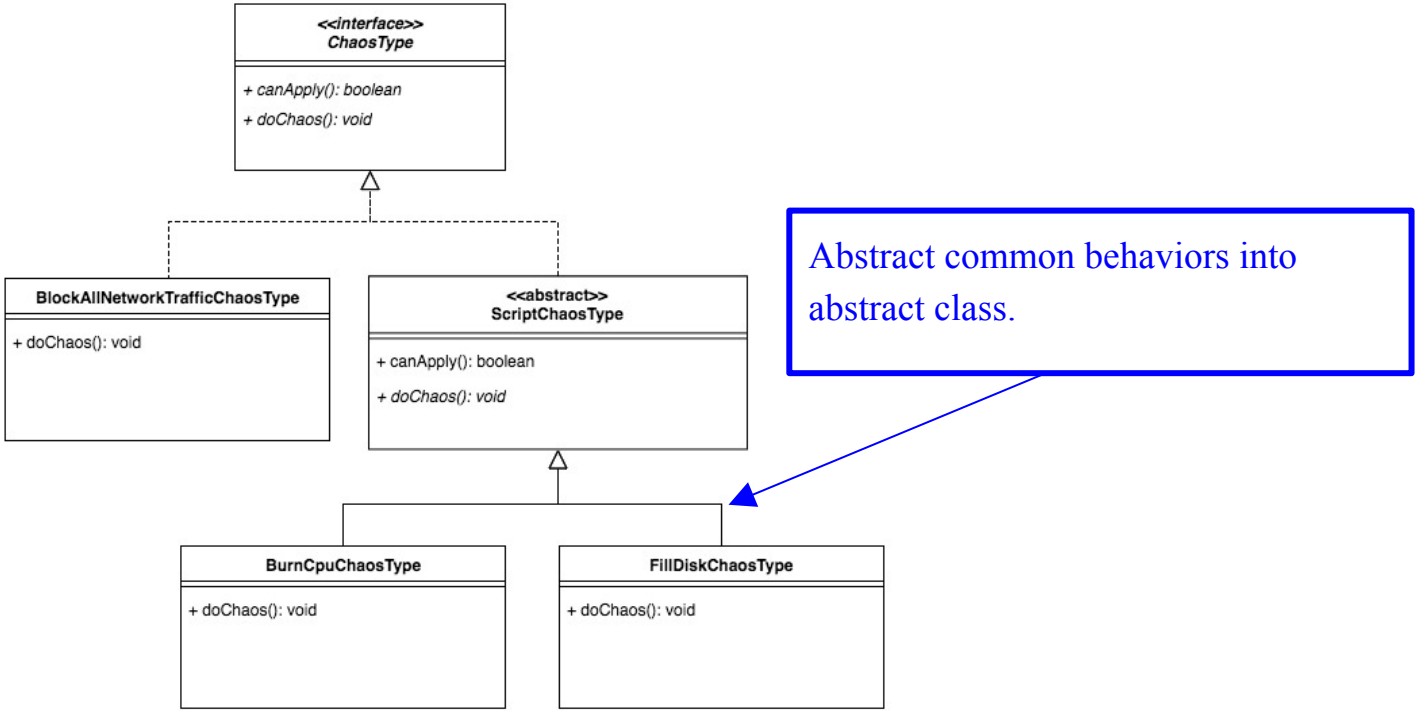


Act-2: Abstract Common Behaviors

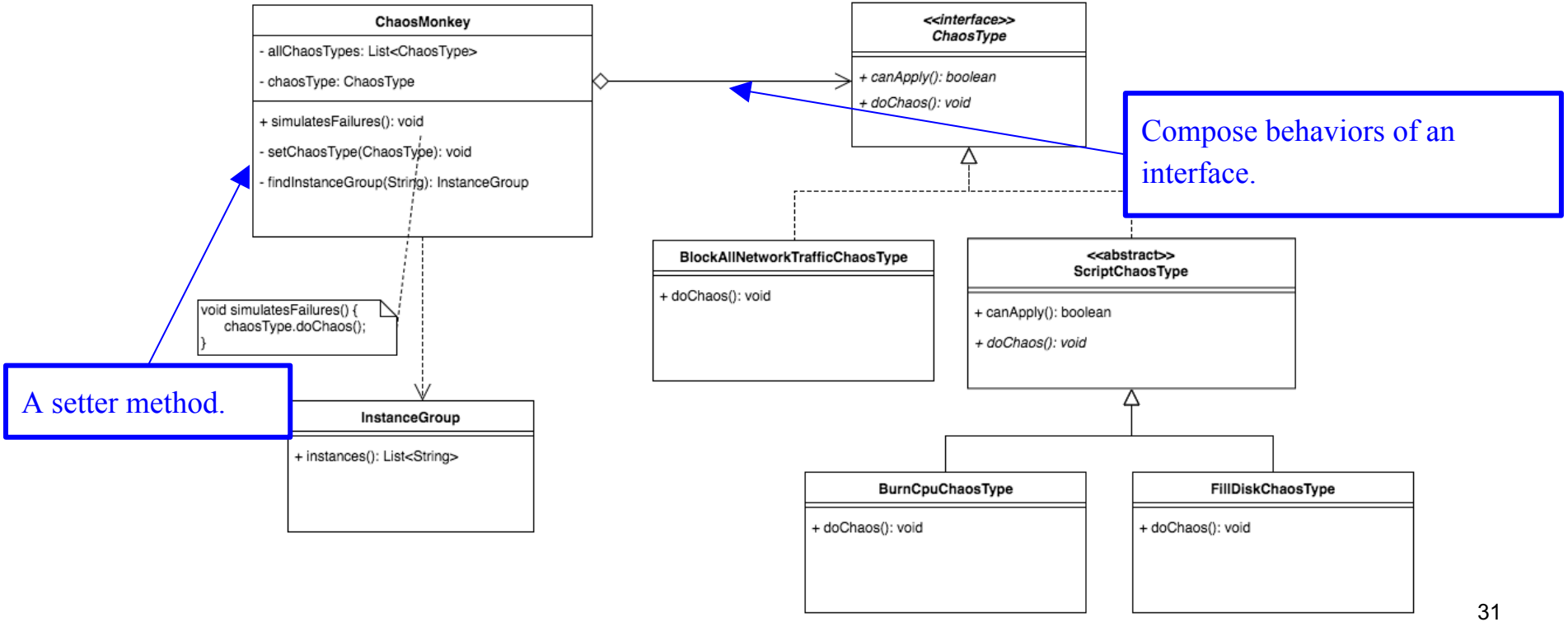
Abstract common behaviors with a same signature into interface through polymorphism.



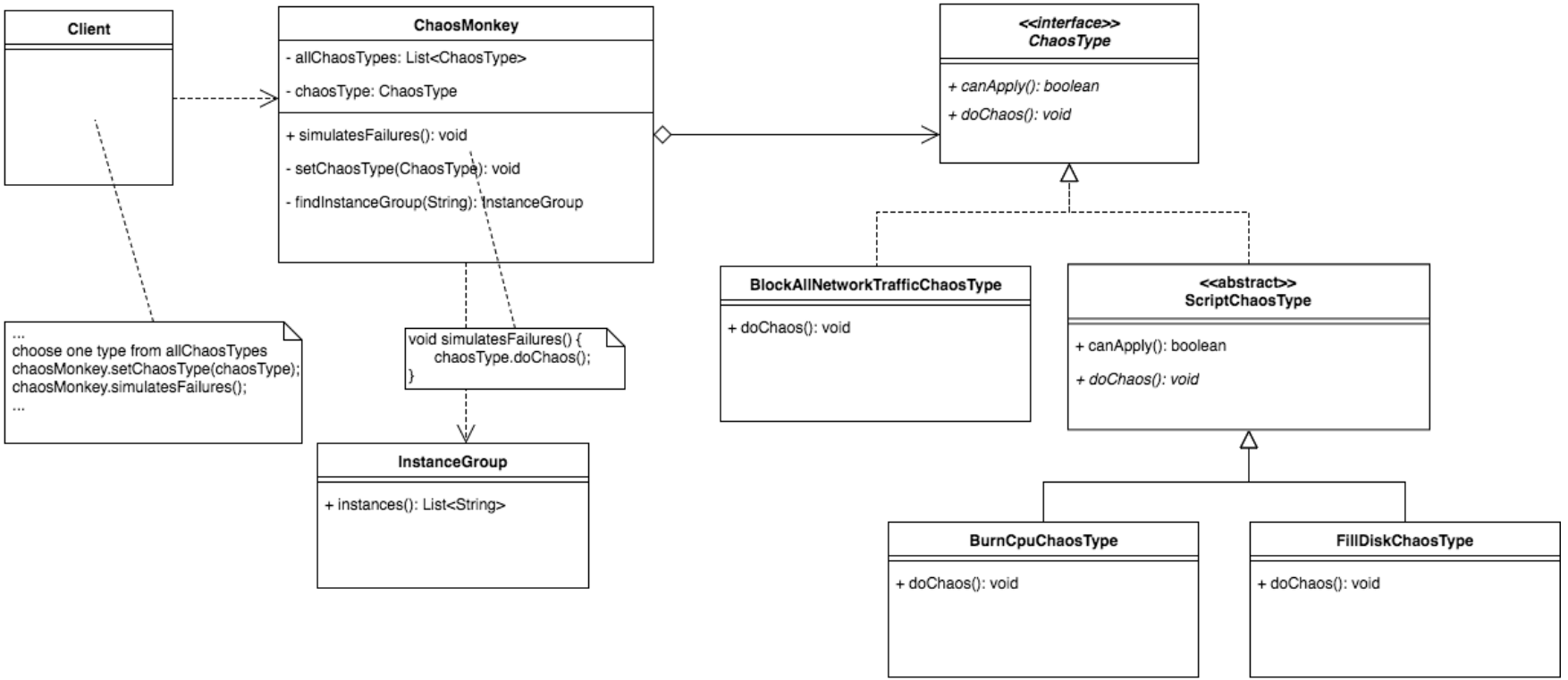
Act-2: Abstract Common Behaviors



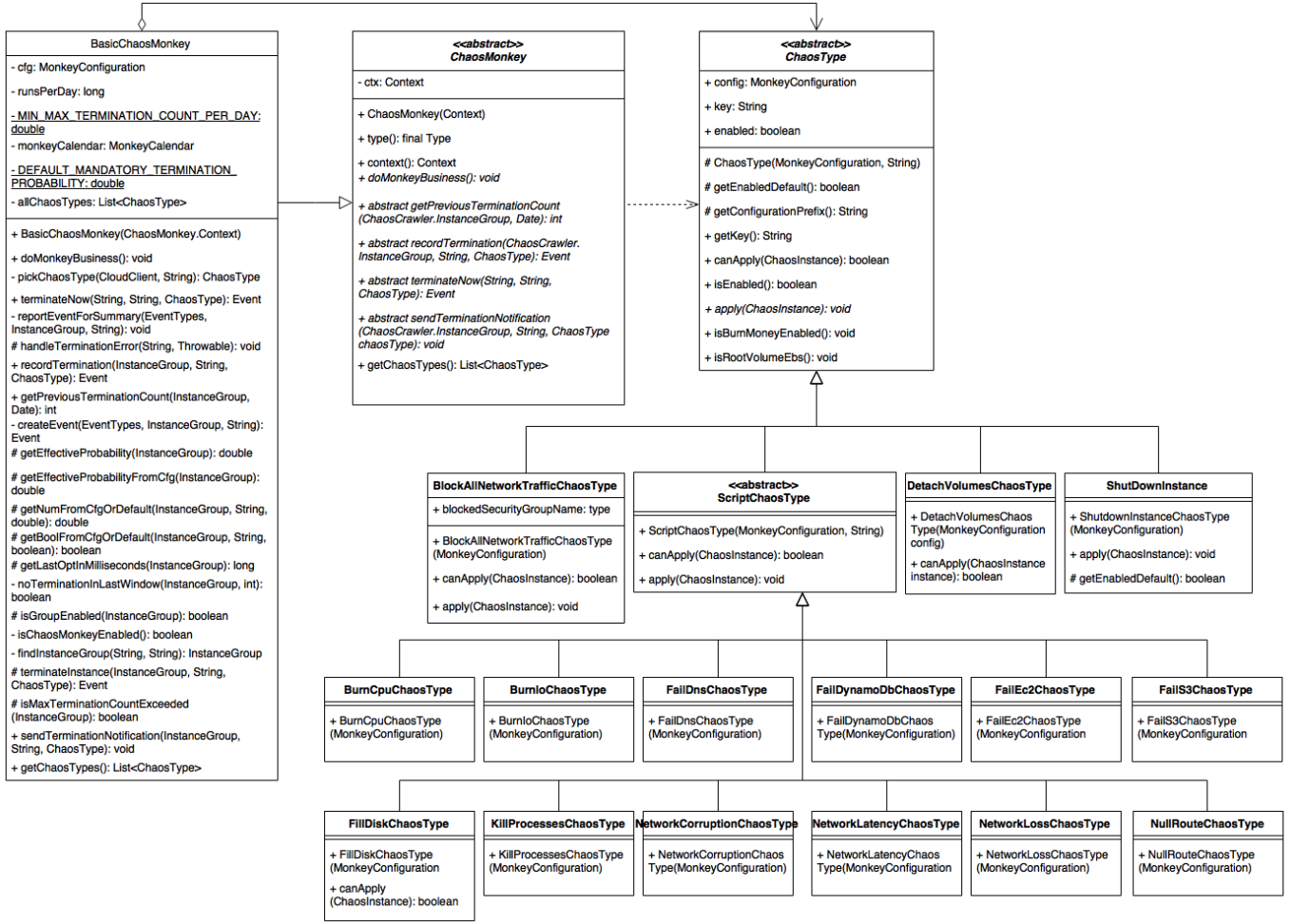
Act-3: Compose Abstract Behaviors

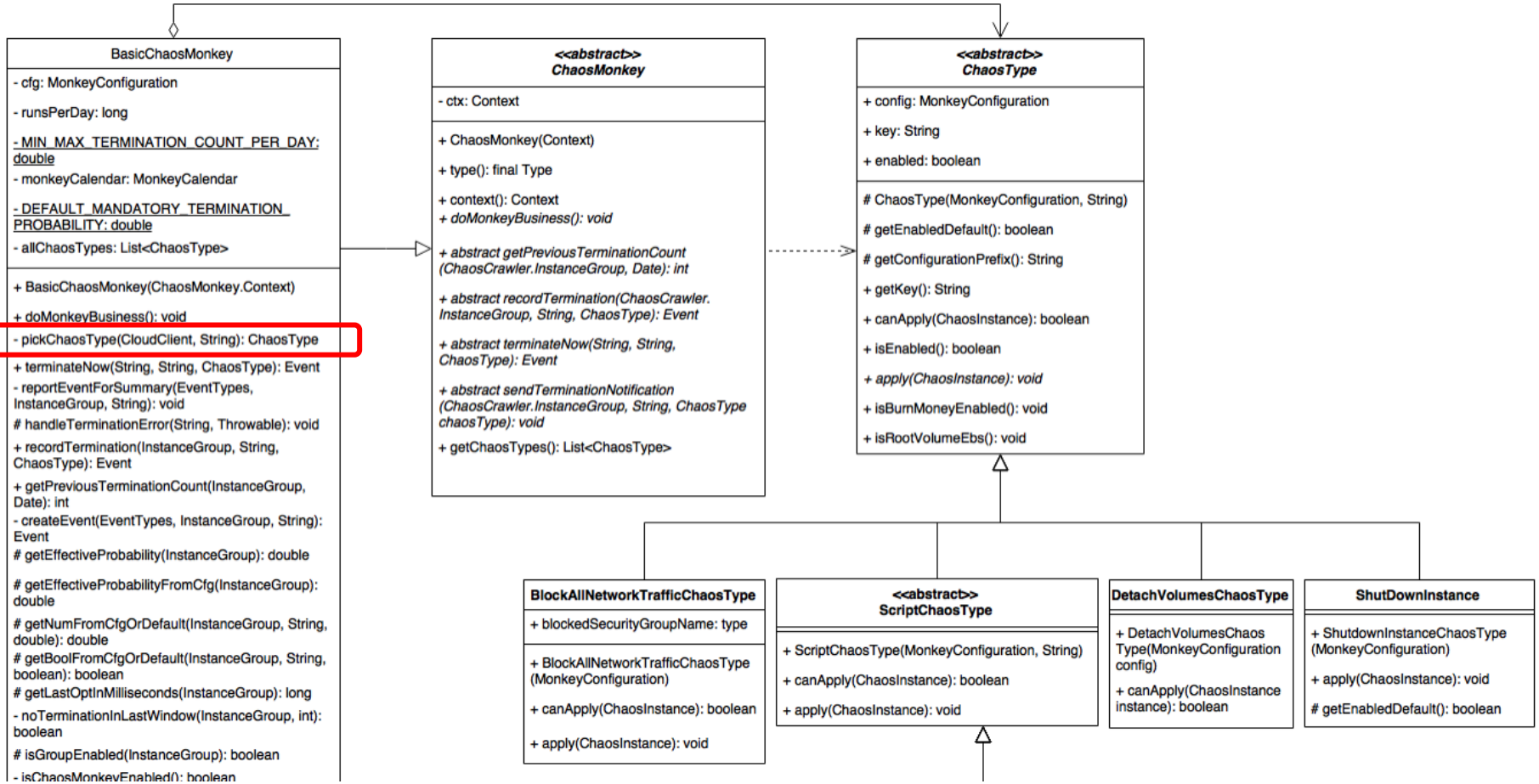


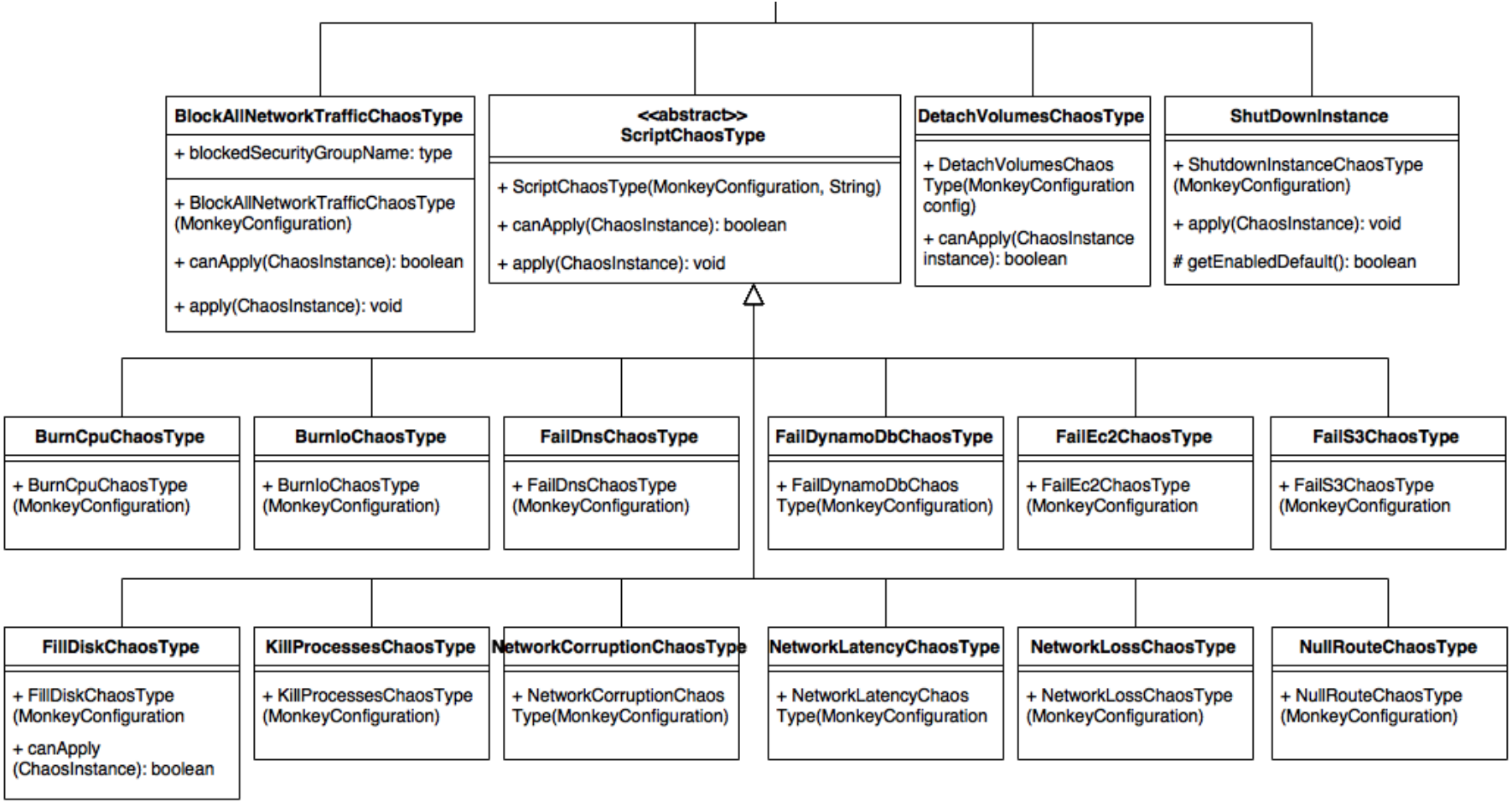
Refactored Design after Design Process



Class Diagram




BasicChaosMonkey
- cfg: MonkeyConfiguration
- runsPerDay: long
- MIN_MAX_TERMINATION_COUNT_PER_DAY: double
- monkeyCalendar: MonkeyCalendar
- DEFAULT_MANDATORY_TERMINATION_PROBABILITY: double
- allChaosTypes: List<ChaosType>
+ BasicChaosMonkey(ChaosMonkey.Context)
+ doMonkeyBusiness(): void
- pickChaosType(CloudClient, String): ChaosType
+ terminateNow(String, String, ChaosType): Event
- reportEventForSummary(EventTypes, InstanceGroup, String): void
handleTerminationError(String, Throwable): void
+ recordTermination(InstanceGroup, String, ChaosType): Event
+ getPreviousTerminationCount(InstanceGroup, Date): int
- createEvent(EventTypes, InstanceGroup, String): Event
getEffectiveProbability(InstanceGroup): double
getEffectiveProbabilityFromCfg(InstanceGroup): double
getNumFromCfgOrDefault(InstanceGroup, String, double): double
getBoolFromCfgOrDefault(InstanceGroup, String, boolean): boolean
getLastOptInMilliseconds(InstanceGroup): long
- noTerminationInLastWindow(InstanceGroup, int): boolean
isGroupEnabled(InstanceGroup): boolean
- isChaosMonkeyEnabled(): boolean
<<abstract>>
ChaosMonkey
- ctx: Context
+ ChaosMonkey(Context)
+ type(): final Type
+ context(): Context
+ doMonkeyBusiness(): void
+ abstract getPreviousTerminationCount (ChaosCrawler.InstanceGroup, Date): int
+ abstract recordTermination(ChaosCrawler. InstanceGroup, String, ChaosType): Event
+ abstract terminateNow(String, String, ChaosType): Event
+ abstract sendTerminationNotification (ChaosCrawler.InstanceGroup, String, ChaosType chaosType): void
+ getChaosTypes(): List<ChaosType>
<<abstract>>
ChaosType
+ config: MonkeyConfiguration
+ key: String
+ enabled: boolean
# ChaosType(MonkeyConfiguration, String)
# getEnabledDefault(): boolean
# getConfigurationPrefix(): String
+ getKey(): String
+ canApply(ChaosInstance): boolean
+ isEnabled(): boolean
+ apply(ChaosInstance): void
+ isBurnMoneyEnabled(): void
+ isRootVolumeEbs(): void
BlockAllNetworkTrafficChaosType
+ blockedSecurityGroupName: type
+ BlockAllNetworkTrafficChaosType (MonkeyConfiguration)
+ canApply(ChaosInstance): boolean
+ apply(ChaosInstance): void
<<abstract>>
ScriptChaosType
+ ScriptChaosType(MonkeyConfiguration, String)
+ canApply(ChaosInstance): boolean
+ apply(ChaosInstance): void
DetachVolumesChaosType
+ DetachVolumesChaos Type(MonkeyConfiguration config)
+ canApply(ChaosInstance instance): boolean
ShutDownInstance
+ ShutdownInstanceChaosType (MonkeyConfiguration)
+ apply(ChaosInstance): void
# getEnabledDefault(): boolean


BlockAllNetworkTrafficChaosType
+ blockedSecurityGroupName: type
+ BlockAllNetworkTrafficChaosType (MonkeyConfiguration)
+ canApply(ChaosInstance): boolean
+ apply(ChaosInstance): void
<<abstract>>
ScriptChaosType
+ ScriptChaosType(MonkeyConfiguration, String)
+ canApply(ChaosInstance): boolean
+ apply(ChaosInstance): void
DetachVolumesChaosType
+ DetachVolumesChaos Type(MonkeyConfiguration config)
+ canApply(ChaosInstance instance): boolean
ShutDownInstance
+ ShutdownInstanceChaosType (MonkeyConfiguration)
+ apply(ChaosInstance): void
# getEnabledDefault(): boolean
BurnCpuChaosType
+ BurnCpuChaosType (MonkeyConfiguration)
BurnIoChaosType
+ BurnIoChaosType (MonkeyConfiguration)
FailDnsChaosType
+ FailDnsChaosType (MonkeyConfiguration)
FailDynamoDbChaosType
+ FailDynamoDbChaos Type(MonkeyConfiguration)
FailEc2ChaosType
+ FailEc2ChaosType (MonkeyConfiguration
FailS3ChaosType
+ FailS3ChaosType (MonkeyConfiguration
FillDiskChaosType
+ FillDiskChaosType (MonkeyConfiguration
+ canApply (ChaosInstance): boolean
KillProcessesChaosType
+ KillProcessesChaosType (MonkeyConfiguration)
NetworkCorruptionChaosType
+ NetworkCorruptionChaos Type(MonkeyConfiguration)
NetworkLatencyChaosType
+ NetworkLatencyChaos Type(MonkeyConfiguration
NetworkLossChaosType
+ NetworkLossChaosType (MonkeyConfiguration)
NullRouteChaosType
+ NullRouteChaosType (MonkeyConfiguration)

Conformity Monkey

Outline

- Introduction
- Requirements for Conformity Monkey
- Initial Design
- Refactored Design from Simian Army Developers
- Problems
  - How to use custom rules in Conformity Monkey
  - How to replace Auto Scaling Groups

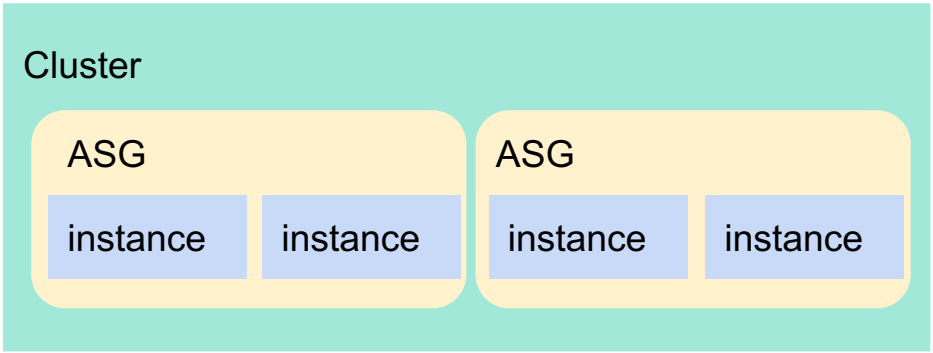
Introduction

- What is Conformity Monkey?
  - A service running on AWS cloud.
  - Provides a conformity check.
- Why is Conformity Monkey required?
  - Check if the applications or instances are launched according to the conforming rules.
- How Conformity Monkey works?
  - Marks and then Notifies.

Requirements for Conformity Monkey$_1$

- Conformity Monkey determines whether an instance in clusters is nonconforming by applying a set of rules on it. If any of the rules determines that the instance is not conforming, the monkey marks the instance and sends an email notification to the owner of the cluster.

- Cluster is the basic unit of conformity check. It includes a single ASG or a group of ASGs and an ASG includes one or many instances. Each cluster has an owner.



Requirements for Conformity Monkey$_2$

Rule **InstanceIsHealthyInEureka** checks if all instances in the cluster are healthy in Discovery.

Rule **InstanceHasHealthCheckUrl** checks if all instances in a cluster has health check url in Discovery/Eureka.

Rule **InstanceHasStatusUrl** checks if all instances in a cluster has status url.

Rule **InstanceInSecurityGroup** checks whether or not all instances in a cluster are in specific security groups.

Rule **InstanceTooOld** checks if there are instances that are older than certain days. Instances are not considered to be permanent in the cloud, so sometimes having too old instances could indicate potential issues.

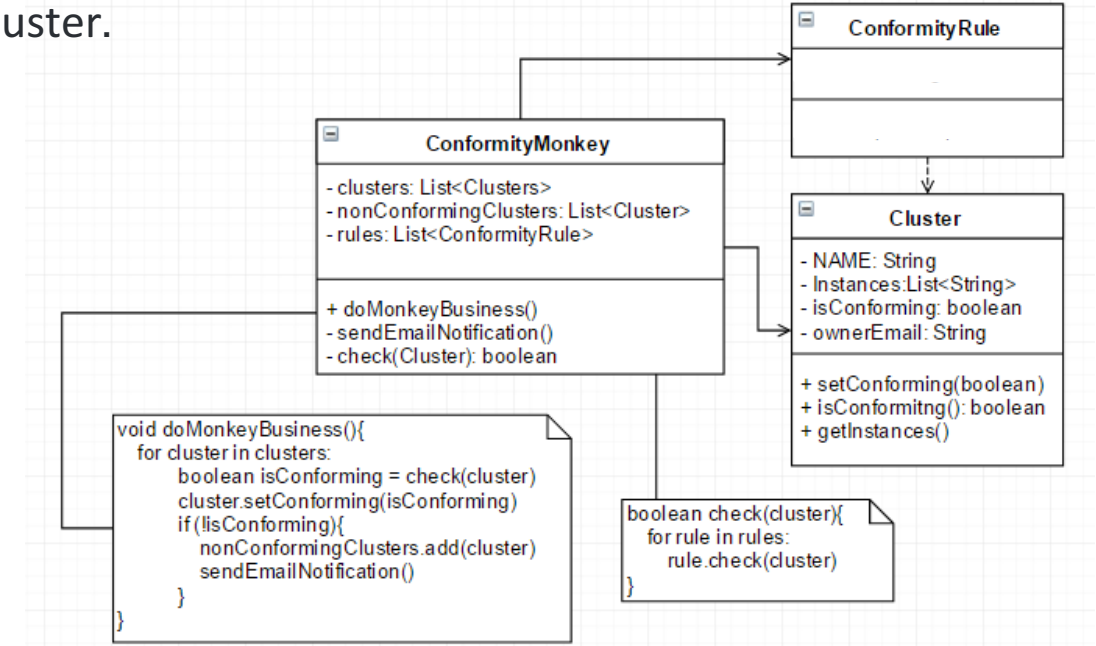
Rule **SameZonesInElbAndAsg** checks if the zones in ELB and ASG are the same.

Rule **InstanceInVPC** checks if an instance (a SoloInstance or an instance in ASG) is in a virtual private cloud.

Rule **CrossZoneLoadBalancing** checks if the cross-zone load balancing is enabled for all cluster ELBs (Elastic Load Balancer).

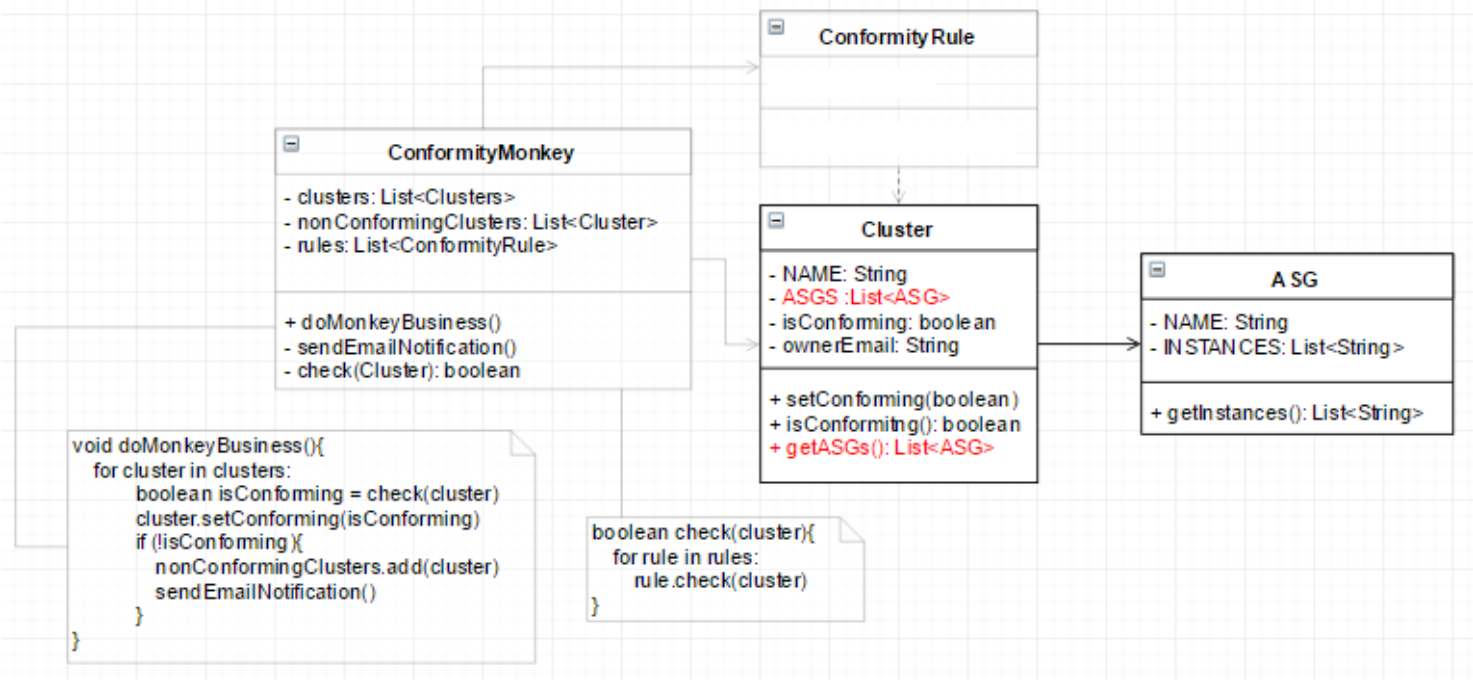
Requirements$_1$

Conformity Monkey determines whether an instance in clusters is nonconforming by applying a set of rules on it. If any of the rules determines that the instance is not conforming, the monkey marks the instance and sends an email notification to the owner of the cluster.



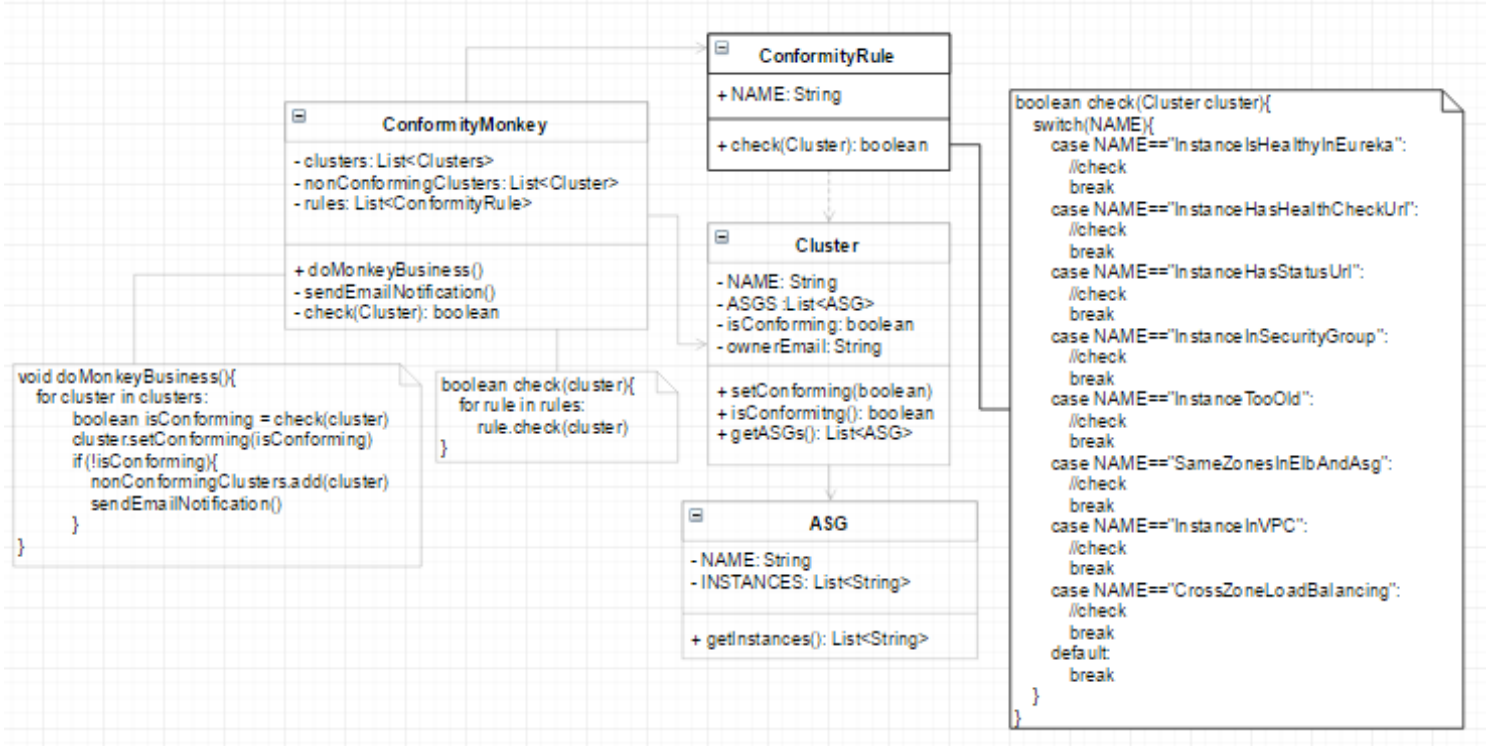
Requirements$_2$

Cluster is the basic unit of conformity check. It includes a single ASG or a group of ASGs and an ASG includes one or many instances. Each cluster has an owner.



Requirements$_3$

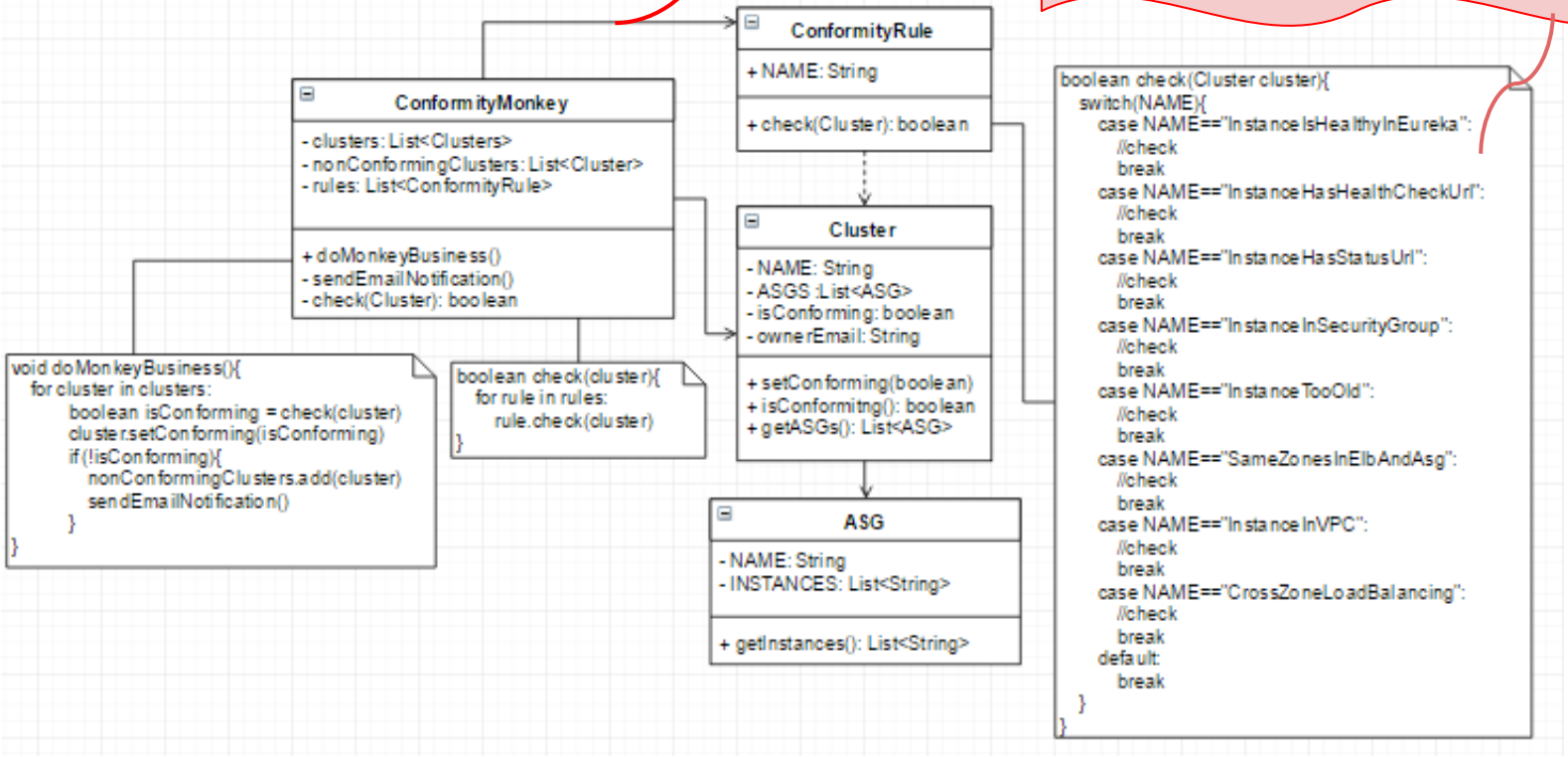
Each rule has its name and its checking mechanism



Initial Design

Violates
Dependency Inversion Principle

Violates
Single Responsibility Principle
&
Open/Closed Principle



Refactor Design

Each rule has their own check mechanism.

The design of Conformity Monkey has to make it simple to customize rules or to add new ones.

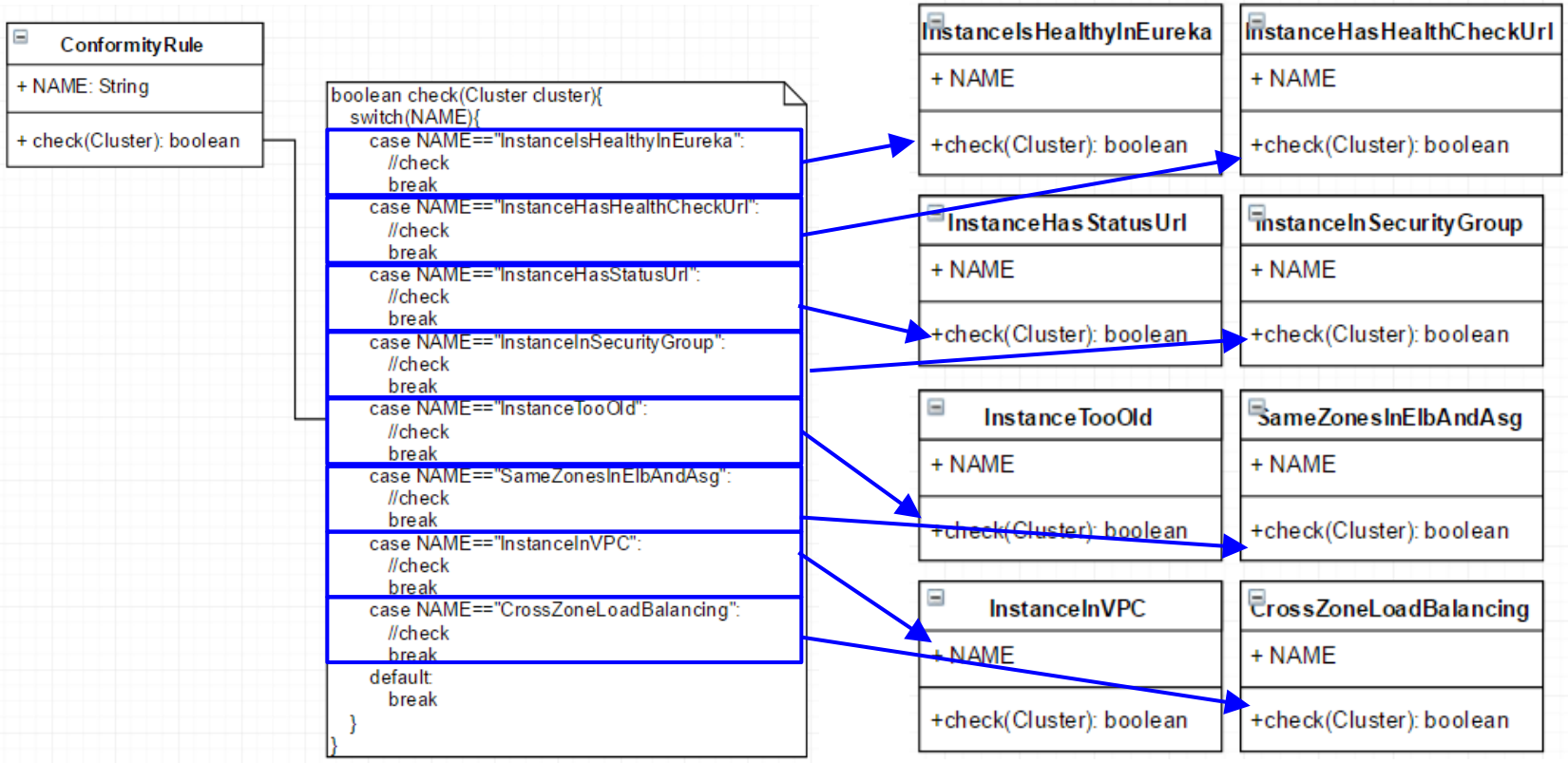
Single Responsibility Principle:
Separate out the rules from the rules processing logic.

Open/Closed Principle:
Add new rules without changing the rest of the system.

Dependency Inversion Principle:
Program to an interface, not an implementation.

Act-1: Encapsulate what varies

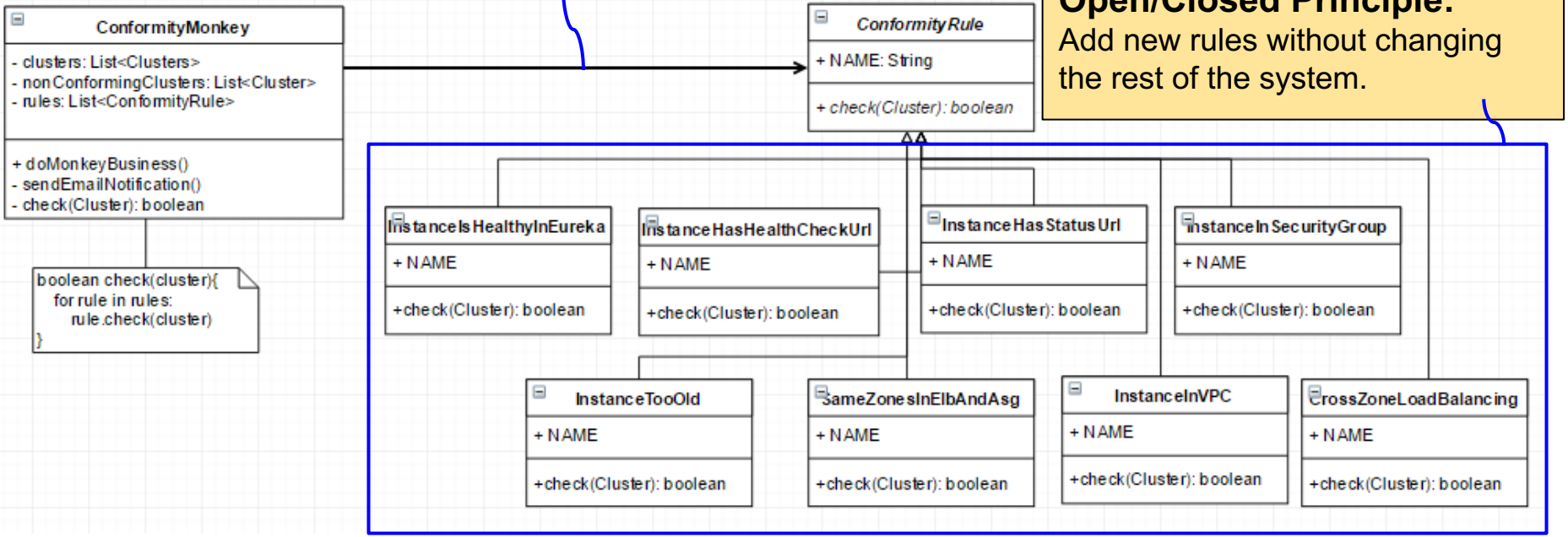
Single Responsibility Principle:
Separate out the rules from the rules processing logic.



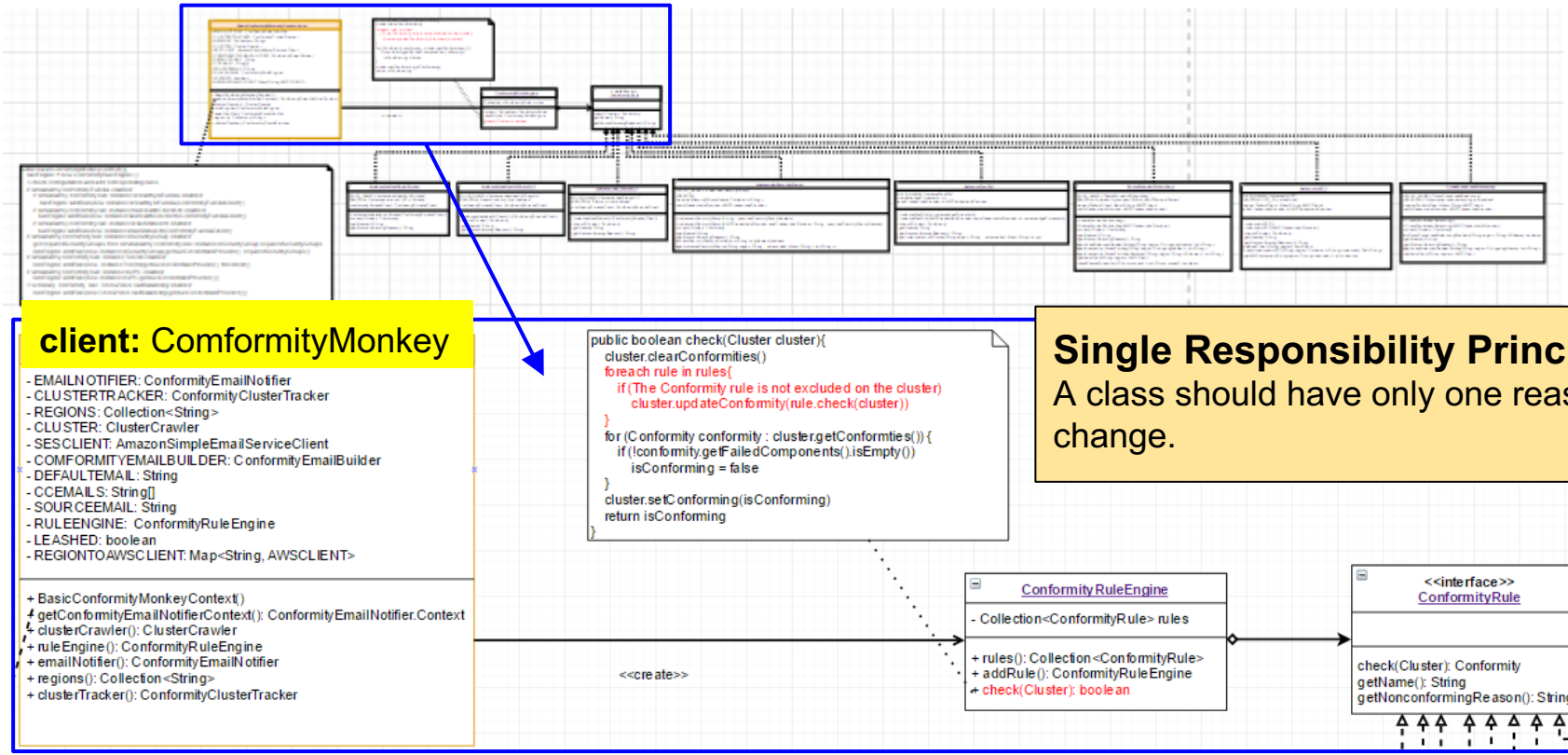
Act-2: Abstract Common Behaviors

Dependency Inversion Principle:
Program to an interface, not an implementation.

Open/Closed Principle:
Add new rules without changing the rest of the system.



Act-3: Use a rule engine as an evaluator



Single Responsibility Principle:
A class should have only one reason to change.

Advantages of using Rule Engine

- Hides data and implementation details from client.
- If a rule has to be checked dynamically according to the current state, there will be an if-else chain in the client; it is difficult to manage the rules.
- With the rule engine, we can encapsulate the operations of the rules. Therefore, the client only interacts with the rule engine to do conformity check.

Final Class Diagram of Conformity Monkey

Designed by Developers of SimianArmy

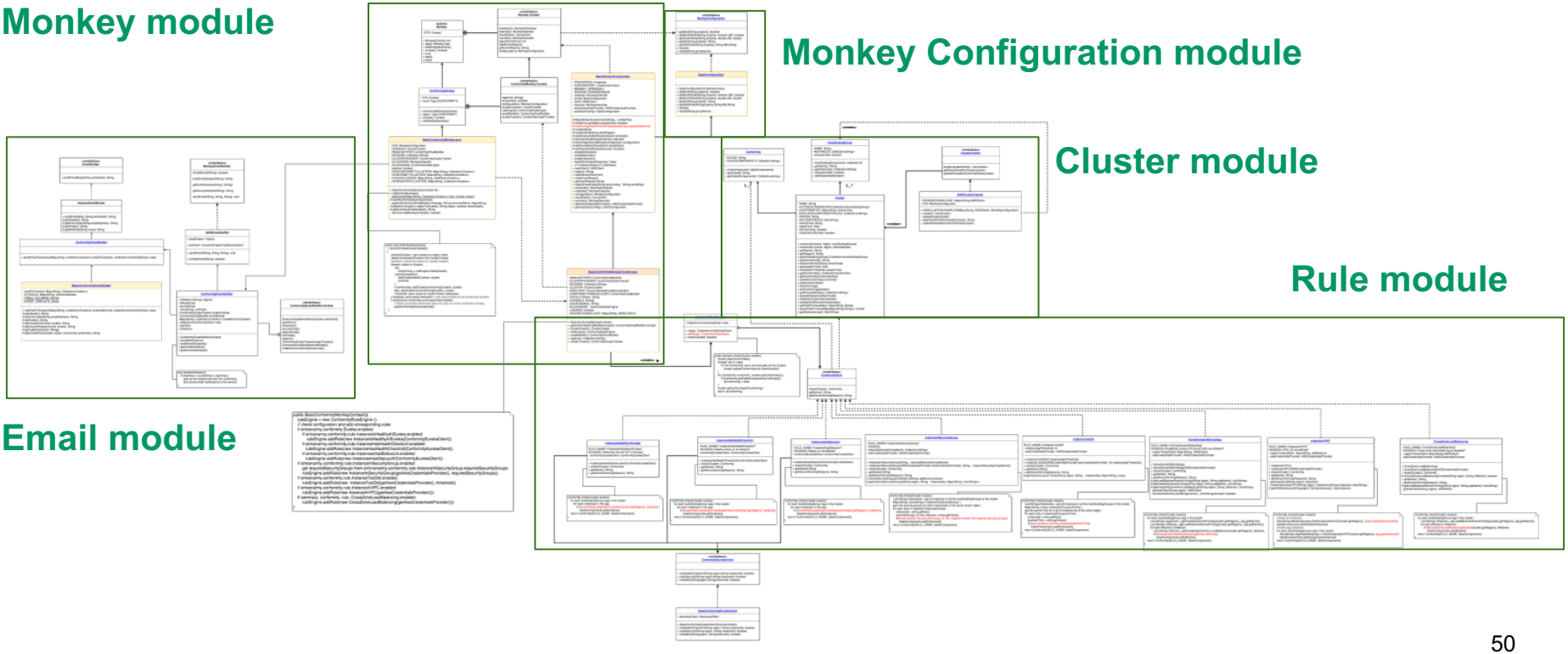
Monkey module

Monkey Configuration module

Cluster module

Rule module

Email module



Problem-1: How to create custom rules for Conformity Monkey

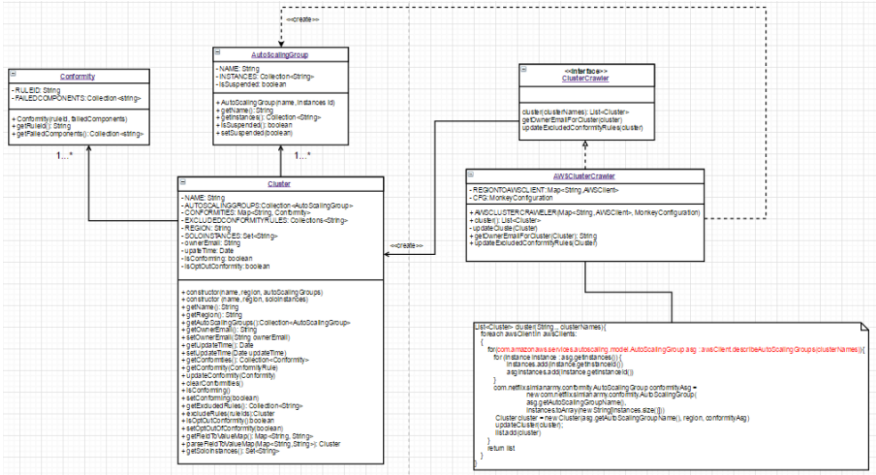
- Create a concrete class that implements **ConformityRule**.
- Add properties in **conformity.properties** to configure and enable/disable the new rule.
- Modify client, **BasicConformityMonkeyContext**, to include the new rule in Monkey Runs.

Problem-2: How to replace Auto Scaling Groups

(from: https://groups.google.com/forum/#!topic/simianarmy-users/4l1veoxU3NE)

Q: Replace ASG with a third-party tool to handle scaling of my EC2 instances .

A: They assume the structure of aws cluster which contains ASGs which contains instances. If you want, you can fork it and modify the iteration over instances to be in a way that doesn't use ASGs.



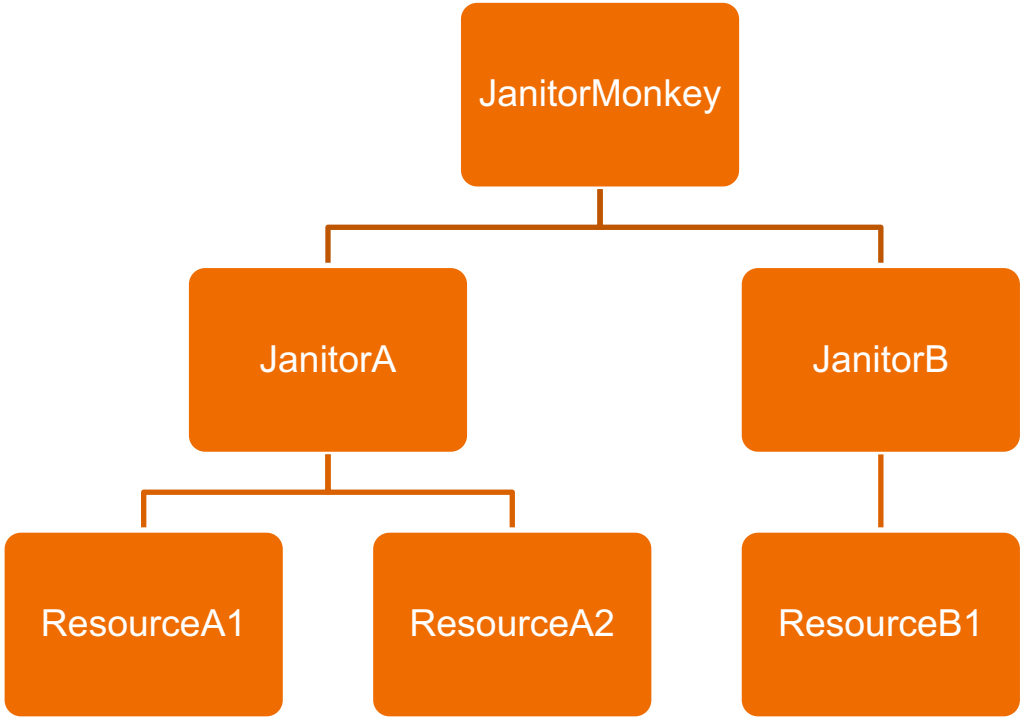
Janitor Monkey

What does Janitor Monkey Do

- Automatically cleans up the resources that are no longer needed.

1. Mark the resources
2. Notify the owners
3. Cleanup the resources

Hierarchy


JanitorMonkey
JanitorA
JanitorB
ResourceA1
ResourceA2
ResourceB1

Janitor Monkey

- Opt in resources
- Opt out resources

Mark a resource

- JanitorRuleEngine

- Rule
- Exclusion Rule

Abstract Janitor

- markResources()
- cleanupResources()

- markedResources
- unmarkedResources
- cleanedResources
- failToCleanResources

Abstract Janitor – flexibility

- Leashed: boolean
- JanitorResourceTracker

```
if (!leashed) {
    resourceTracker.addOrUpdate(resource);
    if (recorder != null) {
        Event evt = recorder.newEvent(
                Type.JANITOR, EventTypes.UNMARK_RESOURCE, resource, resource.getId());
        recorder.recordEvent(evt);
    }
} else {
    LOGGER.info(String.format(
            "The janitor is leashed, no data change is made for unmarking the resource %s.",
            resource.getId()));
}
unmarkedResources.add(resource);
```

Abstract Janitor – abstract operation

- *postMark*()
- *cleanup*()
- *postCleanup()*

Class Diagram

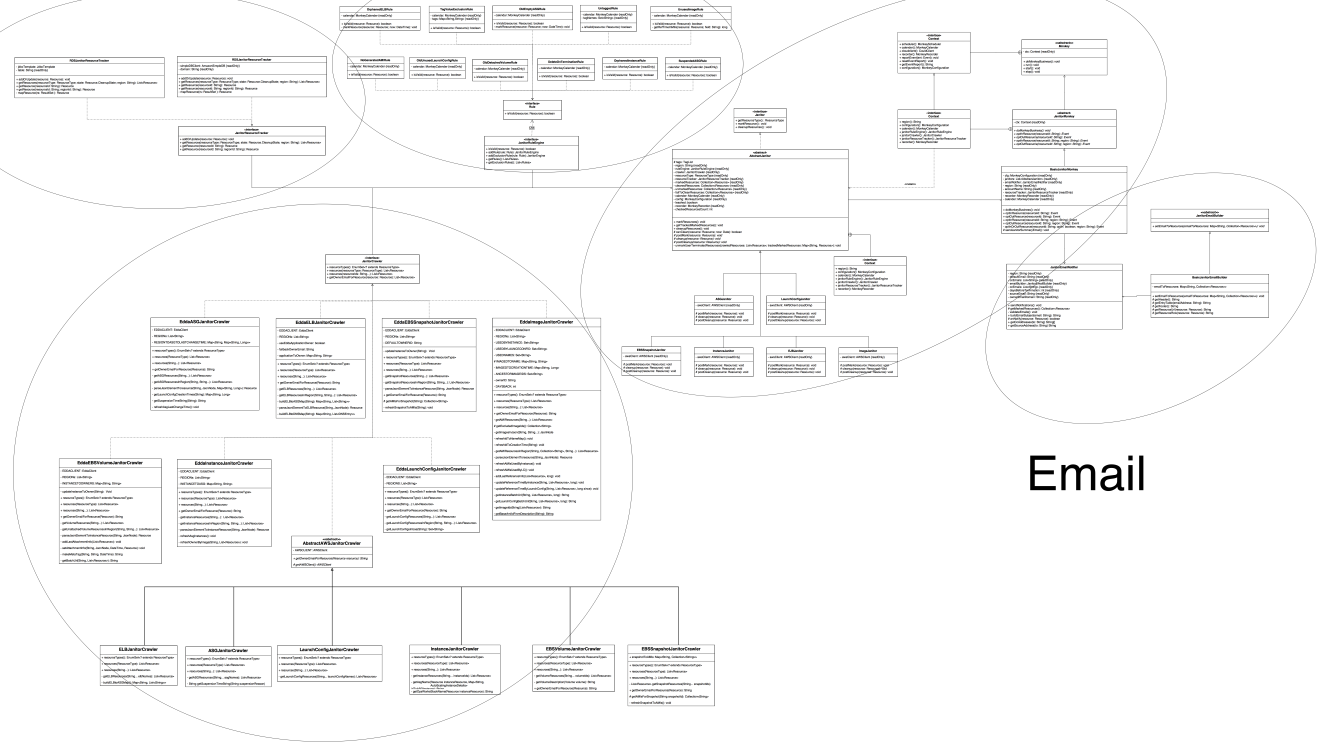
ResourceTracker

RuleEngine

MainBody

Crawler

Email



Client

Introduction$_1$

For those who need to interact with **AWS cloud**.

a. **CloudClient** : Provides the **interface** for the monkeys.(To terminate the instances, delete ASGs, open an ssh connection etc.)

b. **AWSClient** : Implements CloudClient, being a client interface for Simple Amazon **EC2** and Amazon **ASG**.

c. **VSphereClient** : Extends *AWSClient*, Terminate the VMs with the configured *TerminationStrategy*

Introduction$_2$

For monkeys who want to talk to *RESTful web services*.

a. **MonkeyRestClient** : A REST client which can get response in *JSON* and get the base url of the service.

b. **EddaClient** : Extends MonkeyRestClient, accessing *Edda* to get the history of a cloud resource.

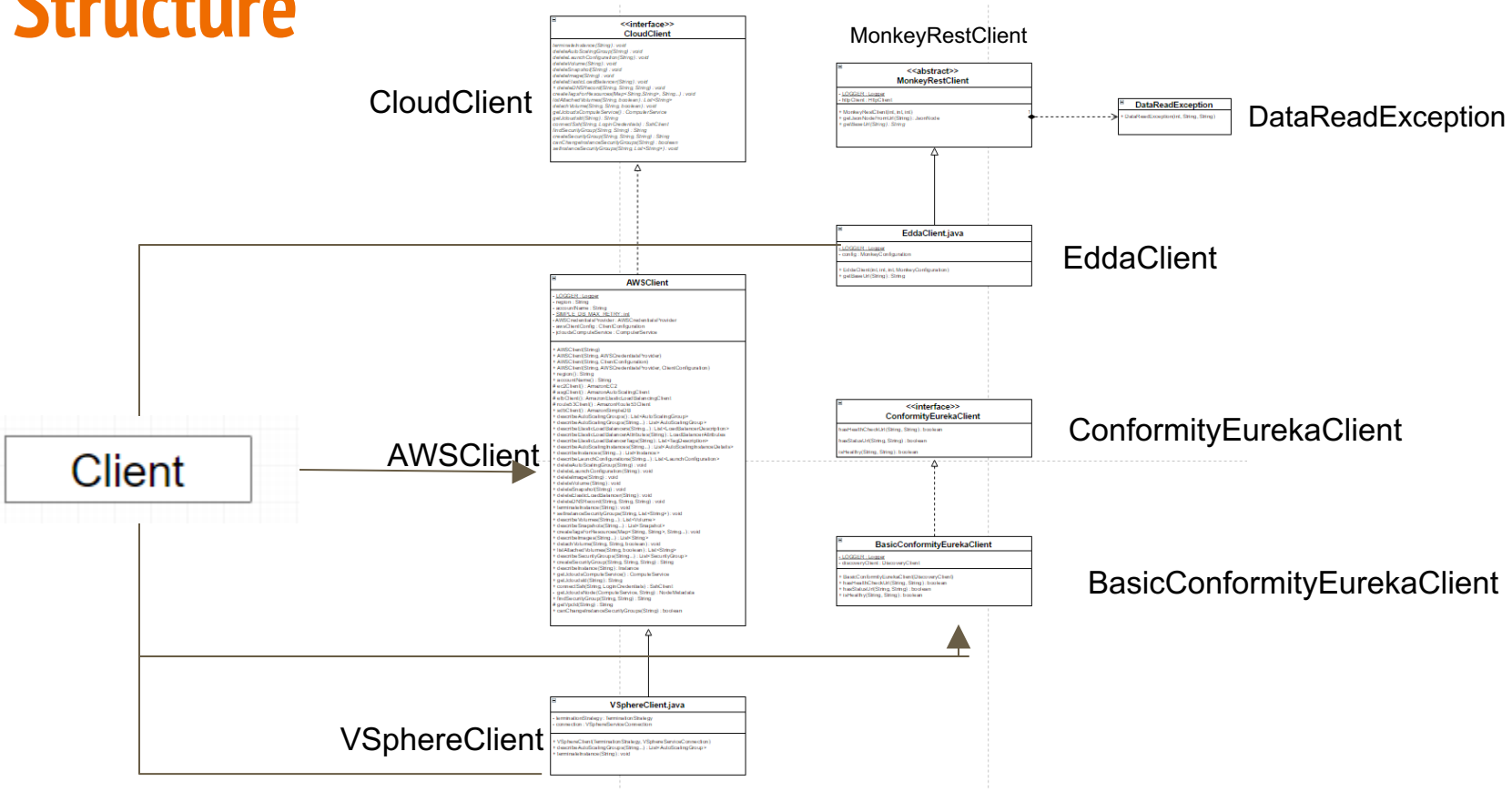
Introduction$_3$

For monkeys who wants to access *Eureka* service.

a. **ConformityEurekaClient** : Provides an interface to get the status of instances for *Conformity Monkey*.

b. **BasicConformityEurekaClient** : Implements a client to access *Eureka* for getting instance information used by *Conformity Monkey*.

Structure


CloudClient
MonkeyRestClient
DataReadException
EddaClient
Client
AWSClient
ConformityEurekaClient
BasicConformityEurekaClient
VSphereClient

Multi-Thread –
ways to synchronize

AWSClient

```
private synchronized void init() {
    if (nextId != null) {
        return;
    }
    File dbFile = null;
    dbFile = (dbFilename == null) ? tempDbFile() : new File(dbFilename);
    if (dbpassword != null) {
        db = DBMaker.newFileDB(dbFile)
                .closeOnJvmShutdown()
                .encryptionEnable(dbpassword)
                .make();
    } else {
        db = DBMaker.newFileDB(dbFile)
                .closeOnJvmShutdown()
                .make();
    }
    eventMap = db.getTreeMap("eventMap");
    nextId = db.createAtomicLong("next", 1);
}
```

Initial implementation.

```
private void init() {
    if (nextId == null) {
        synchronized(this) {
            if (nextId == null) {
                File dbFile = null;
                dbFile = (dbFilename == null) ? tempDbFile() : new File(dbFilename);
                if (dbpassword != null) {
                    db = DBMaker.newFileDB(dbFile)
                            .closeOnJvmShutdown()
                            .encryptionEnable(dbpassword)
                            .make();
                } else {
                    db = DBMaker.newFileDB(dbFile)
                            .closeOnJvmShutdown()
                            .make();
                }
                eventMap = db.getTreeMap("eventMap");
                nextId = db.createAtomicLong("next", 1);
            }
        }
    }
}
```

Refactored implementation.

LocalDbRecorder

```
/** {@inheritDoc} */
@Override
public synchronized ComputeService getJcloudsComputeService() {
    if (jcloudsComputeService == null) {
        String username = awsCredentialsProvider.getCredentials().getAWSAccessKeyId();
        String password = awsCredentialsProvider.getCredentials().getAWSSecretKey();
        ComputeServiceContext jcloudsContext = ContextBuilder.newBuilder("aws-ec2").credentials(username, password)
                .modules(ImmutableSet.<Module>of(new SLF4JLoggingModule(), new JschSshClientModule()))
                .buildView(ComputeServiceContext.class);

        this.jcloudsComputeService = jcloudsContext.getComputeService();
    }

    return jcloudsComputeService;
}
```

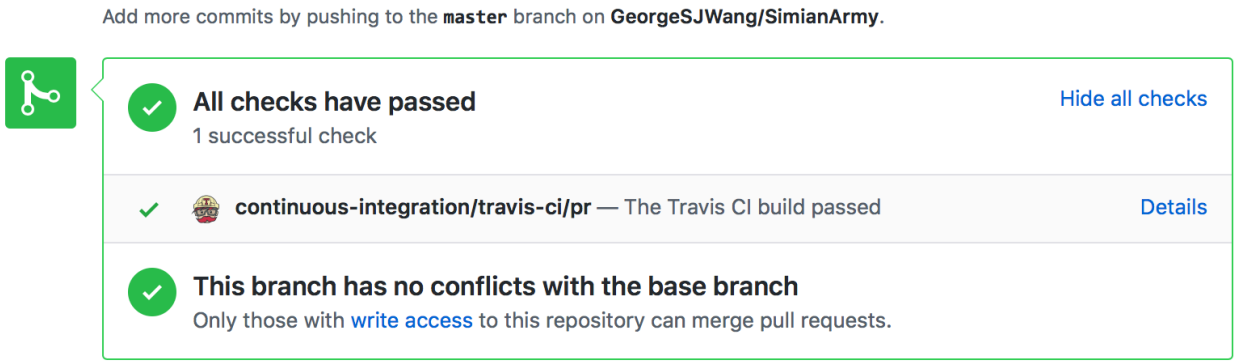
Initial implementation.

```
/** {@inheritDoc} */
@Override
public ComputeService getJcloudsComputeService() {
    if (jcloudsComputeService == null) {
        synchronized(this) {
            if (jcloudsComputeService == null) {
                String username = awsCredentialsProvider.getCredentials().getAWSAccessKeyId();
                String password = awsCredentialsProvider.getCredentials().getAWSSecretKey();
                ComputeServiceContext jcloudsContext = ContextBuilder.newBuilder("aws-ec2").credentials(username, password)
                        .modules(ImmutableSet.<Module>of(new SLF4JLoggingModule(), new JschSshClientModule()))
                        .buildView(ComputeServiceContext.class);

                this.jcloudsComputeService = jcloudsContext.getComputeService();
            }
        }
    }

    return jcloudsComputeService;
}
```

Refactored implementation.

GitHub Pull Request

- Pull request was submitted last Thursday.
- It is now pending and waiting to be merged.

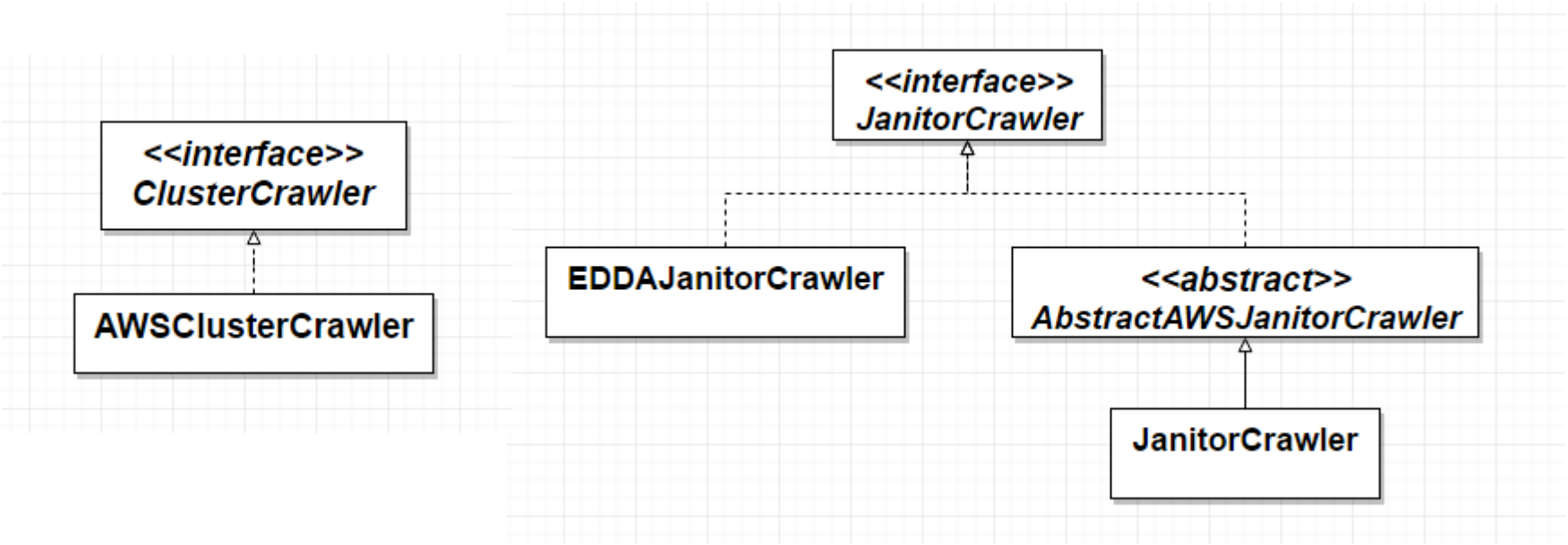


Crawler

Objective

- Use AWS API or EDDA API to query data by either “ID” or “Region” from cloud server, and transform them into List<Cluster> or List<Resource> type for Conformity monkey and Janitor monkey to use.

Simplified Architecture



How to achieve?

- AWS-based :
- take InstanceJanitorCrawler as an example
- EDDA-based:
- take EDDAInstanceJanitorCrawler as an example

AWS-Based

```
public List<Instance> describeInstances(String... instanceIds) {
    if (instanceIds == null || instanceIds.length == 0) {
        LOGGER.info(String.format("Getting all EC2 instances in region %s.", region));
    } else {
        LOGGER.info(String.format("Getting EC2 instances for %d ids in region %s.", instanceIds.length, region));
    }

    List<Instance> instances = new LinkedList<Instance>();

    AmazonEC2 ec2Client = ec2Client();
    DescribeInstancesRequest request = new DescribeInstancesRequest();
    if (instanceIds != null) {
        request.withInstanceIds(Arrays.asList(instanceIds));
    }
    DescribeInstancesResult result = ec2Client.describeInstances(request);
    for (Reservation reservation : result.getReservations()) {
        instances.addAll(reservation.getInstances());
    }

    LOGGER.info(String.format("Got %d EC2 instances in region %s.", instances.size(), region));
    return instances;
}
```

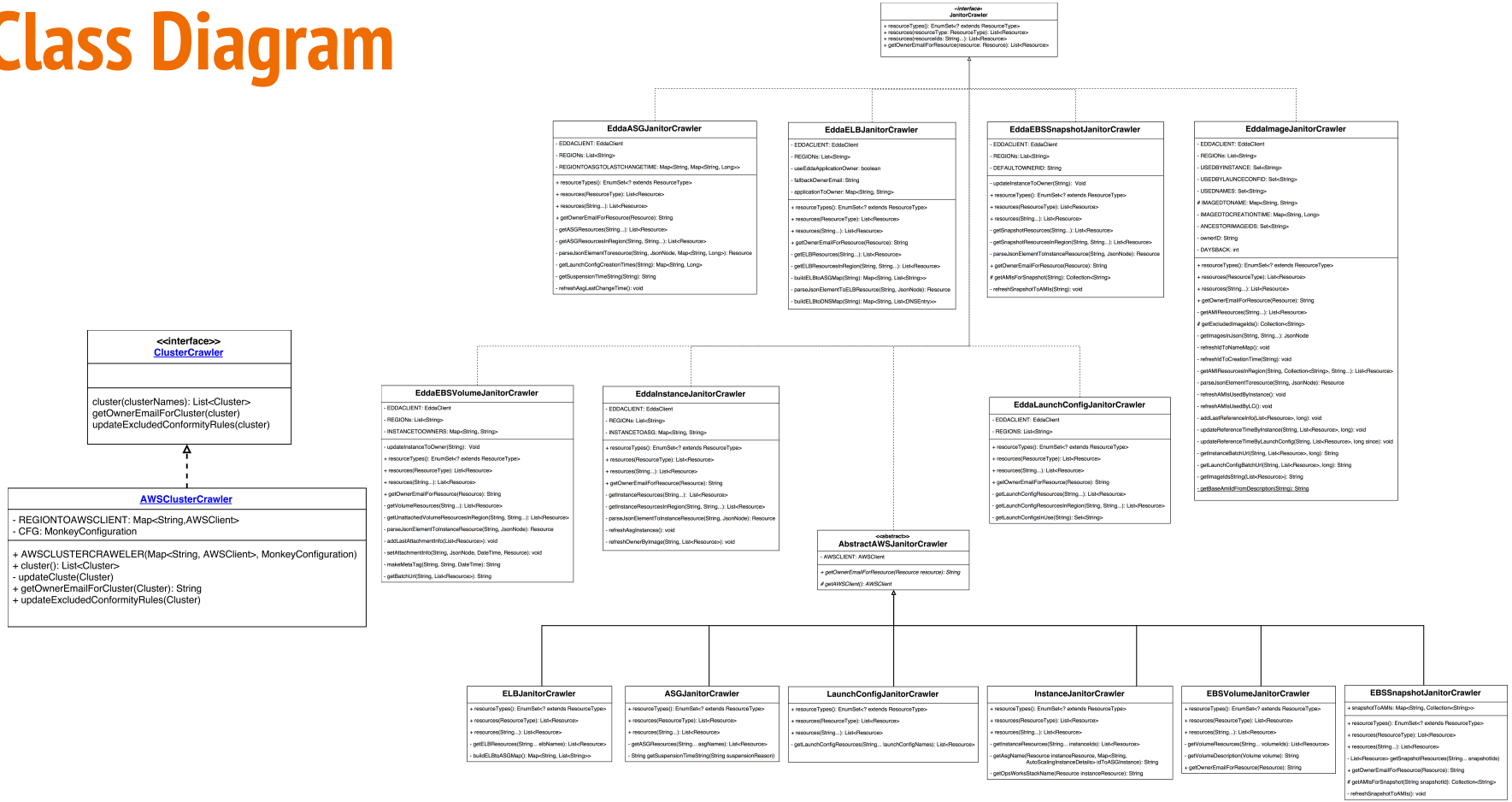
```
private List<Resource> getInstanceResources(String... instanceIds) {
    List<Resource> resources = new LinkedList<Resource>();

    AWSClient awsClient = getAWSClient();
    Map<String, AutoScalingInstanceDetails> idToASGInstance = new HashMap<String, AutoScalingInstanceDetails>();
    for (AutoScalingInstanceDetails instanceDetails : awsClient.describeAutoScalingInstances(instanceIds)) {
        idToASGInstance.put(instanceDetails.getInstanceId(), instanceDetails);
    }

    for (Instance instance : awsClient.describeInstances(instanceIds)) {
        Resource instanceResource = new AWSResource().withId(instance.getInstanceId())
                .withRegion(getAWSClient().region()).withResourceType(AWSResourceType.INSTANCE)
                .withLaunchTime(instance.getLaunchTime());
        for (Tag tag : instance.getTags()) {
            instanceResource.setTag(tag.getKey(), tag.getValue());
        }
        String description = String.format("type=%s; host=%s", instance.getInstanceType(),
                instance.getPublicDnsName() == null ? "" : instance.getPublicDnsName());
        instanceResource.setDescription(description);
        instanceResource.setOwnerEmail(getOwnerEmailForResource(instanceResource));

        String asgName = getAsgName(instanceResource, idToASGInstance);
        if (asgName != null) {
            instanceResource.setAdditionalField(INSTANCE_FIELD_ASG_NAME, asgName);
            LOGGER.info(String.format("instance %s has a ASG tag name %s.", instanceResource.getId(), asgName));
        }
        String opsworksStackName = getOpsWorksStackName(instanceResource);
        if (opsworksStackName != null) {
            instanceResource.setAdditionalField(INSTANCE_FIELD_OPSWORKS_STACK_NAME, opsworksStackName);
            LOGGER.info(String.format("instance %s is part of an OpsWorks stack named %s.", instanceResource.getId(), opsworksStackName));
        }
        if (instance.getState() != null) {
            ((AWSResource) instanceResource).setAWSResourceState(instance.getState().getName());
        }
        resources.add(instanceResource);
    }
    return resources;
}
```

EDDA-Based

```
public JsonNode getJsonNodeFromUrl(String url) throws IOException {
    LOGGER.info(String.format("Getting Json response from url: %s", url));
    HttpGet request = new HttpGet(url);
    request.setHeader("Accept", "application/json");
    HttpResponse response = httpClient.execute(request);

    InputStream is = response.getEntity().getContent();
    String jsonContent;
    if (is != null) {
        Scanner s = new Scanner(is, "UTF-8").useDelimiter("\\A");
        jsonContent = s.hasNext() ? s.next() : "";
        is.close();
    } else {
        return null;
    }

    int code = response.getStatusLine().getStatusCode();
    if (code == 404) {
        return null;
    } else if (code >= 300 || code < 200) {
        throw new DataReadException(code, url, jsonContent);
    }

    JsonNode result;
    try {
        ObjectMapper mapper = new ObjectMapper();
        result = mapper.readTree(jsonContent);
    } catch (Exception e) {
        throw new RuntimeException(String.format("Error trying to parse json response from url %s, got: %s",
                url, jsonContent), e);
    }
    return result;
}
```

```
private List<Resource> getLaunchConfigResourcesInRegion(String region, String... launchConfigNames) {
    String url = eddaClient.getBaseUrl(region) + "/aws/launchConfigurations;";
    if (launchConfigNames != null && launchConfigNames.length != 0) {
        url += StringUtils.join(launchConfigNames, ',');
        LOGGER.info(String.format("Getting launch configurations in region %s for %d ids",
                region, launchConfigNames.length));
    } else {
        LOGGER.info(String.format("Getting all launch configurations in region %s", region));
    }
    url += ";_expand:(launchConfigurationName,createdTime)";

    JsonNode jsonNode = null;
    try {
        jsonNode = eddaClient.getJsonNodeFromUrl(url);
    } catch (Exception e) {
        LOGGER.error(String.format(
                "Failed to get Jason node from edda for instances in region %s.", region), e);
    }

    if (jsonNode == null || !jsonNode.isArray()) {
        throw new RuntimeException(String.format("Failed to get valid document from %s, got: %s", url, jsonNode));
    }

    List<Resource> resources = Lists.newArrayList();

    Set<String> usedLCs = getLaunchConfigsInUse(region);

    for (Iterator<JsonNode> it = jsonNode.getElements(); it.hasNext();) {
        JsonNode launchConfiguration = it.next();
        String lcName = launchConfiguration.get("launchConfigurationName").getTextValue();
        Resource lcResource = new AWSResource().withId(lcName)
                .withRegion(region).withResourceType(AWSResourceType.LAUNCH_CONFIG)
                .withLaunchTime(new Date(launchConfiguration.get("createdTime").getLongValue()));
        lcResource.setOwnerEmail(getOwnerEmailForResource(lcResource));

        lcResource.setAdditionalField(LAUNCH_CONFIG_FIELD_USED_BY_ASG, String.valueOf(usedLCs.contains(lcName)));
        resources.add(lcResource);
    }
    return resources;
}
```

Class Diagram



Issue$_1$

- Why does each cloud service need a corresponding crawler?
- Any way to increase code reusability?

Response$_1$

- Each crawler has similar functionality, but with different parameters, descriptions, additional fields, tags, Logger messages.

- Take getInstanceResources() and getELBResources() for examples.

```
private List<Resource> getInstanceResources(String... instanceIds) {
    List<Resource> resources = new LinkedList<Resource>();

    AWSClient awsClient = getAWSClient();
    Map<String, AutoScalingInstanceDetails> idToASGInstance = new HashMap<String, AutoScalingInstanceDetails>();
    for (AutoScalingInstanceDetails instanceDetails : awsClient.describeAutoScalingInstances(instanceIds)) {
        idToASGInstance.put(instanceDetails.getInstanceId(), instanceDetails);
    }

    for (Instance instance : awsClient.describeInstances(instanceIds)) {
        Resource instanceResource = new AWSResource().withId(instance.getInstanceId())
                .withRegion(getAWSClient().region()).withResourceType(AWSResourceType.INSTANCE)
                .withLaunchTime(instance.getLaunchTime());
        for (Tag tag : instance.getTags()) {
            instanceResource.setTag(tag.getKey(), tag.getValue());
        }
        String description = String.format("type=%s; host=%s", instance.getInstanceType(),
                instance.getPublicDnsName() == null ? "" : instance.getPublicDnsName());
        instanceResource.setDescription(description);
        instanceResource.setOwnerEmail(getOwnerEmailForResource(instanceResource));

        String asgName = getAsgName(instanceResource, idToASGInstance);
        if (asgName != null) {
            instanceResource.setAdditionalField(INSTANCE_FIELD_ASG_NAME, asgName);
            LOGGER.info(String.format("instance %s has a ASG tag name %s.", instanceResource.getId(), asgName));
        }
        String opsworksStackName = getOpsWorksStackName(instanceResource);
        if (opsworksStackName != null) {
            instanceResource.setAdditionalField(INSTANCE_FIELD_OPSWORKS_STACK_NAME, opsworksStackName);
            LOGGER.info(String.format("instance %s is part of an OpsWorks stack named %s.", instanceResource.getId(), opsworksStackName));
        }
```

```
private List<Resource> getELBResources(String... elbNames) {
    List<Resource> resources = new LinkedList<Resource>();
    AWSClient awsClient = getAWSClient();

    for (LoadBalancerDescription elb : awsClient.describeElasticLoadBalancers(elbNames)) {
        Resource resource = new AWSResource().withId(elb.getLoadBalancerName())
                .withRegion(getAWSClient().region()).withResourceType(AWSResourceType.ELB)
                .withLaunchTime(elb.getCreatedTime());
        resource.setOwnerEmail(getOwnerEmailForResource(resource));
        resources.add(resource);
        List<Instance> instances = elb.getInstances();
        if (instances == null || instances.size() == 0) {
            resource.setAdditionalField("instanceCount", "0");
            resource.setDescription("instances=none");
            LOGGER.debug(String.format("No instances found for ELB %s", resource.getId()));
        } else {
            resource.setAdditionalField("instanceCount", "" + instances.size());
            ArrayList<String> instanceList = new ArrayList<String>(instances.size());
            LOGGER.debug(String.format("Found %d instances for ELB %s", instances.size(), resource.getId()));
            for (Instance instance : instances) {
                String instanceId = instance.getInstanceId();
                instanceList.add(instanceId);
            }
            String instancesStr = StringUtils.join(instanceList,",");
            resource.setDescription(String.format("instances=%s", instances));
            LOGGER.debug(String.format("Resource ELB %s has instances %s", resource.getId(), instancesStr));
        }

        for(TagDescription tagDescription : awsClient.describeElasticLoadBalancerTags(resource.getId())) {
```

Issue$_2$

- Conformity monkey and Janitor monkey both use AWS-based crawler, is it possible to share a common crawler?

Response$_2$

- Conformity monkey uses Cluster as a basic unit, however Janitor monkey uses Resource as a basic unit.

- Both crawlers have similar functionality, but different return datatype, tag, description, Logger message …

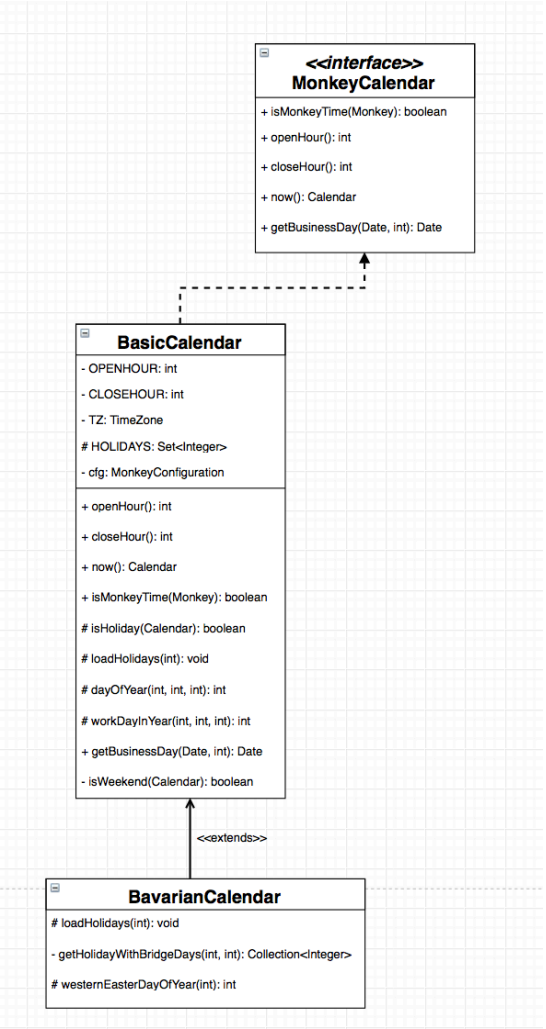
Utilities

Calendar

- Open/Closed Principle
- To add a new type of calendar, such as Chinese Calendar, simply needs extension of BasicCalendar class.
  Hence, not a single java code needs to be modified.

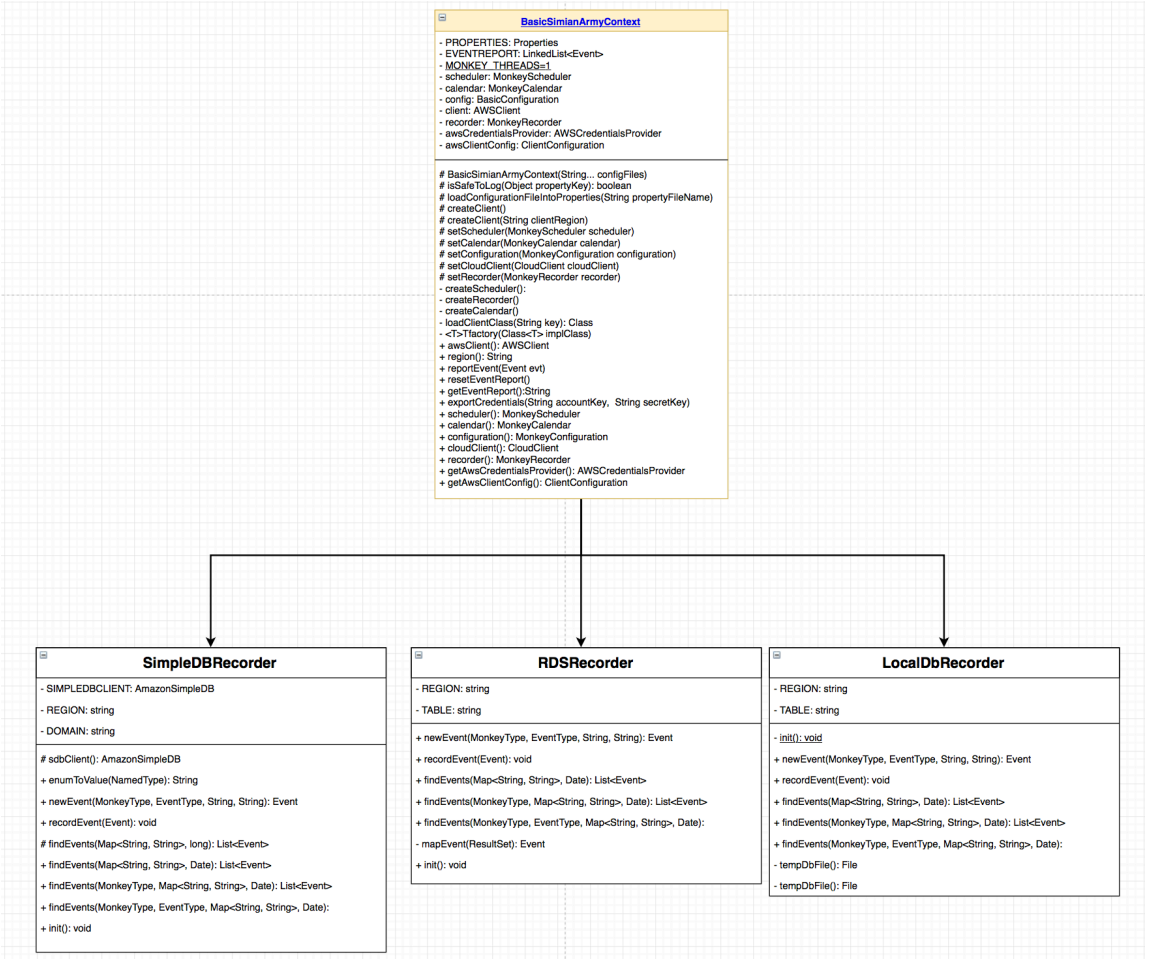
Calendar

- Class diagram:



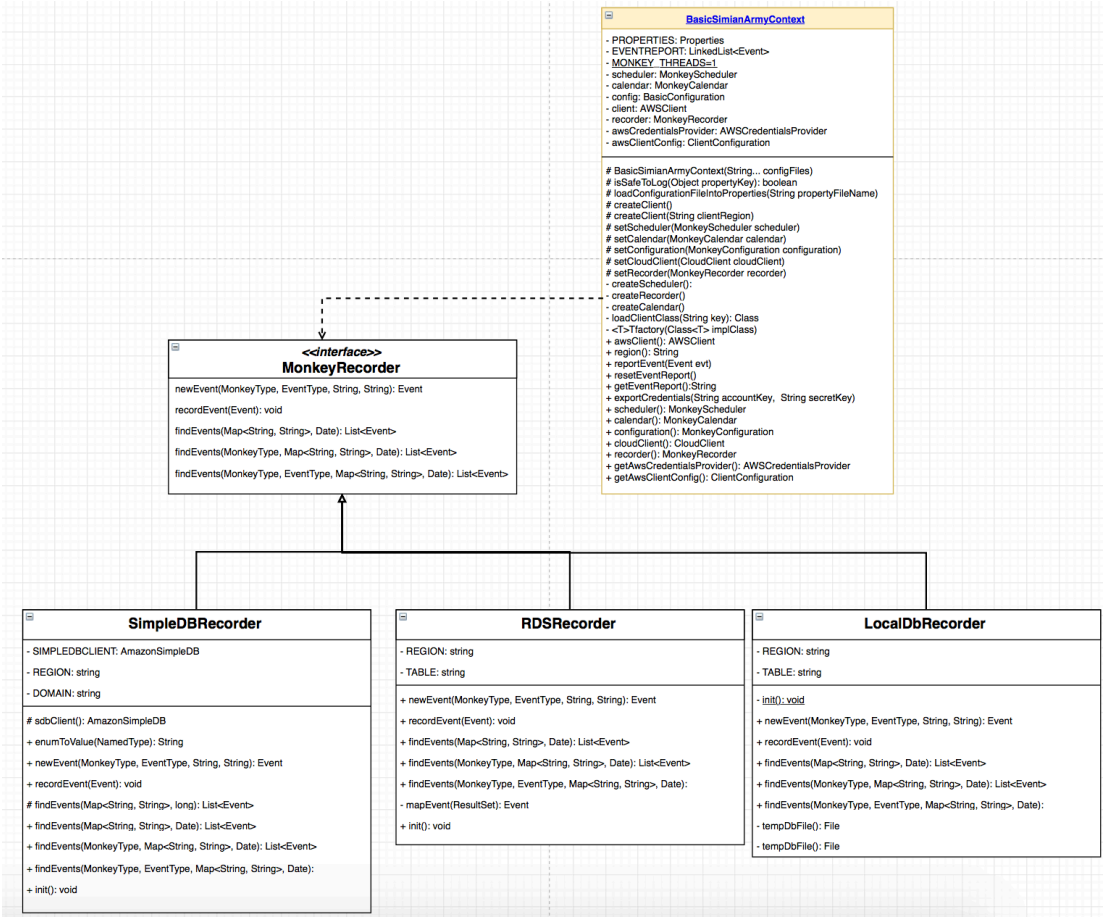
MonkeyRecorder

- Initial Design



MonkeyRecorder

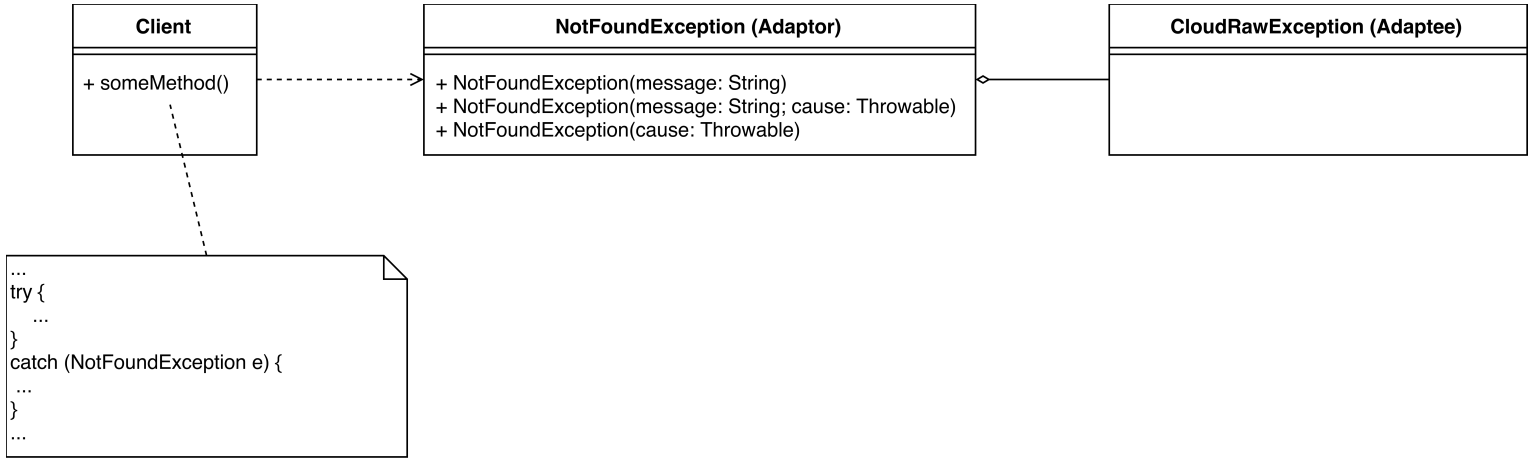
- Dependency Inversion Principle



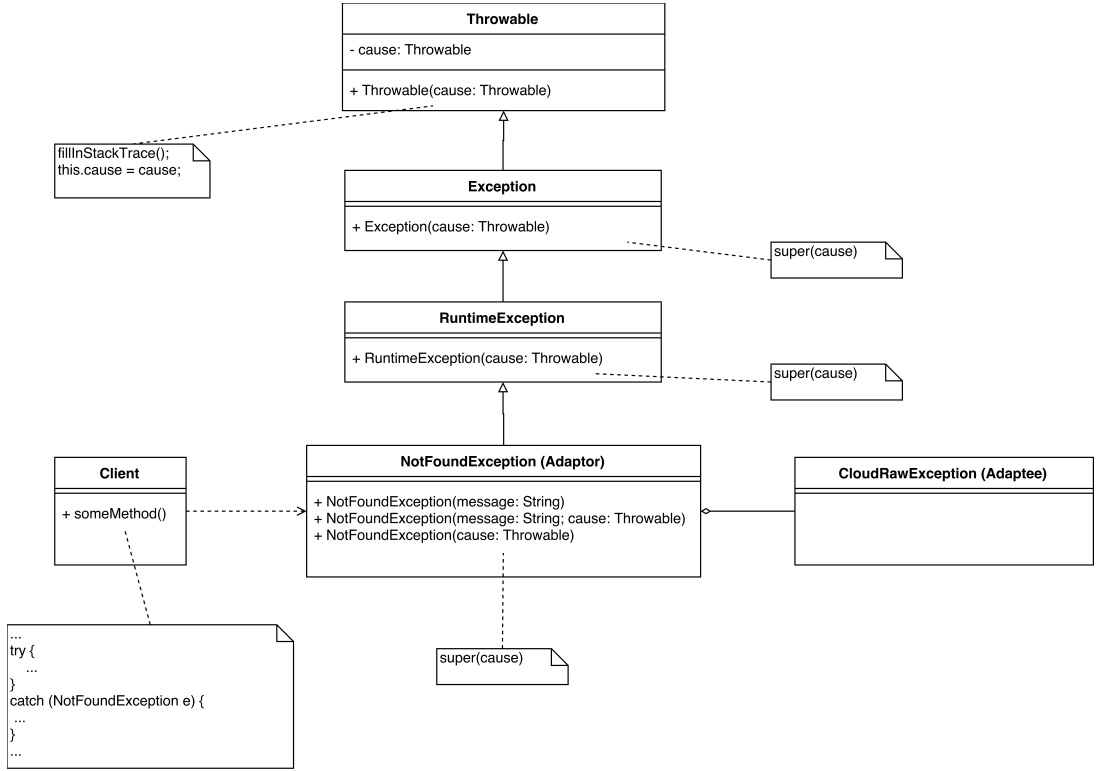
Scenario

- Simian Army defines a new class **NotFoundException**
- **Problem:** Some **outside** exceptions should also be classified as **NotFoundException**
- **Solution:** Use a wrapper to wrap those exceptions

Class Diagram



Class Diagram



Inside SimianArmy

- terminateNow() in BasicChaosMonkey.java

```
if (instances.size() == 1) {
    return terminateInstance(group, instances.iterator().next(), chaosType);
} else {
    throw new NotFoundException(String.format("No instance is found in group %s [type %s]",
            name, type));
}
```

Outside SimianArmy

- detachVolume() in AWSClient.java

```
} catch (AmazonServiceException e) {
    if (e.getErrorCode().equals("InvalidInstanceID.NotFound")) {
        throw new NotFoundException("AWS instance " + instanceId + " not found", e);
    }
    throw e;
}
```

Client

- addTerminationEvent() in ChaosMonkeyResource.java

```
} catch (NotFoundException e) {
    // Available instance cannot be found to terminate, maybe the instance is already gone
    responseStatus = Response.Status.GONE;
    gen.writeStringField("message", e.getMessage());
}
```

- The client only needs to know the exception belongs to **NotFoundException**, the adapter hides all the other details.

Coding Style

TODO Comments

```
private NodeMetadata getJcloudsNode(ComputeService computeService, String jcloudsId) {
    // Work around a jclouds bug / documentation issue...
    // TODO: Figure out what's broken, and eliminate this function

    // This should work (?):
    // Set<NodeMetadata> nodes = computeService.listNodesByIds(Collections.singletonLis

    Set<NodeMetadata> nodes = Sets.newHashSet();
    for (ComputeMetadata n : computeService.listNodes()) {
        if (jcloudsId.equals(n.getId())) {
            nodes.add((NodeMetadata) n);
        }
    }

    if (nodes.isEmpty()) {
        LOGGER.warn("Unable to find jclouds node: {}", jcloudsId);
        for (ComputeMetadata n : computeService.listNodes()) {
            LOGGER.info("Did find node: {}", n);
        }
        throw new IllegalStateException("Unable to find node using jclouds: " + jclouds
    }
    NodeMetadata node = Iterables.getOnlyElement(nodes);
    return node;
}
```

Logger

```
for (AbstractJanitor janitor : janitors) {
    LOGGER.info(String.format("Running %s janitor for region %s", janitor.getResourceType(), janitor.getRegion()));
    try {
            janitor.markResources();
    } catch (Exception e) {
            monkeyErrors.incrementAndGet();
            LOGGER.error(String.format("Got an exception while %s janitor was marking for region %s", janitor.getRes
    }
    LOGGER.info(String.format("Marked %d resources of type %s in the last run.",
            janitor.getMarkedResources().size(), janitor.getResourceType().name()));
    LOGGER.info(String.format("Unmarked %d resources of type %s in the last run.",
            janitor.getUnmarkedResources().size(), janitor.getResourceType()));
}
```

Annotations

```
/** {@inheritDoc} */
@Override
public String findSecurityGroup(String instanceId, String groupName) {
    String vpcId = getVpcId(instanceId);

    SecurityGroup found = null;
    List<SecurityGroup> securityGroups = describeSecurityGroups(vpcId, groupName);
    for (SecurityGroup sg : securityGroups) {
        if (Objects.equal(vpcId, sg.getVpcId())) {
            if (found != null) {
                throw new IllegalStateException("Duplicate security groups found");
            }
            found = sg;
        }
    }
    if (found == null) {
        return null;
    }
    return found.getGroupId();
}
```

Javadoc Comment

```java
import com.netflix.simianarmy.Resource;
import com.netflix.simianarmy.ResourceType;
import com.netflix.simianarmy.aws.AWSResource;
import com.netflix.simianarmy.aws.AWSResourceType;
import com.netflix.simianarmy.client.aws.AWSClient;

/**
 * The crawler to crawl AWS instances for janitor monkey.
 */
public class InstanceJanitorCrawler extends AbstractAWSJanitorCrawler {

    /** The name representing the additional field name of ASG's name. */
    public static final String INSTANCE_FIELD_ASG_NAME = "ASG_NAME";

    /** The name representing the additional field name of the OpsWork stack name. */
    public static final String  INSTANCE_FIELD_OPSWORKS_STACK_NAME = "OPSWORKS_STACK_NAME";

    /** The Constant LOGGER. */
    private static final Logger LOGGER = LoggerFactory.getLogger(InstanceJanitorCrawler.class);

    /**
     * Instantiates a new basic instance crawler.
     * @param awsClient
     *            the aws client
     */
    public InstanceJanitorCrawler(AWSClient awsClient) {
        super(awsClient);
    }
```

Q & A