Extract Method

"Extract Method" is a code refactoring technique that involves breaking down a complex section of code into smaller, more manageable methods or functions.

1. The presented image illustrates an example of "Extract Method".

   After applying the respective refactoring, the code has improved, now having higher quality, being less complex, and easier to understand, analyze, and modify.

Before

```
public static EventType<Void> domMutation(Consumer<DomMutationEvent> handler) {
  Require.nonNull( argName: "Handler", handler);

  URL url = CdpEventTypes.class.getResource( name: "/org/openqa/selenium/devtools/mutation-listener.js");
  if (url == null) {
    throw new IllegalStateException("Unable to find helper script");
  }
  String script;
  try {
    script = Resources.toString(url, UTF_8);
  } catch (IOException e) {
    throw new IllegalStateException("Unable to read helper script");
  }

  return new EventType<Void>() {...};
}
```

After

```
public static EventType<Void> domMutation(Consumer<DomMutationEvent> handler) {
  Require.nonNull( argName: "Handler", handler);

  String script = eventMutation();

  return new EventType<Void>() {...};
}
```

New Method

```
private static String eventMutation() {
  URL url = CdpEventTypes.class.getResource( name: "/org/openqa/selenium/devtools/mutation-listener.js");
  if (url == null) {
    throw new IllegalStateException("Unable to find helper script");
  }
  String script;
  try {
    script = Resources.toString(url, UTF_8);
  } catch (IOException e) {
    throw new IllegalStateException("Unable to read helper script");
  }
  return script;
}
```

1 2 3 4 5

Stro ◯ ◯ ◯ ◯ ◯ Strongly agree

Extract Class

"Extract Class" is a code refactoring technique that involves breaking down a class into smaller, more focused classes.

1. The presented image illustrates an example of "Extract Class".

   After applying the respective refactoring, the code has improved, now having higher quality, being less complex, and easier to understand, analyze, and modify.

Before

```
public class BitmapTransitionOptions {
  @NonNull
  public BitmapTransitionOptions crossFade(
      @NonNull DrawableCrossFadeFactory drawableCrossFadeFactory) {
    return transitionUsing(drawableCrossFadeFactory);
  }

  @NonNull
  public static BitmapTransitionOptions withWrapped(
      @NonNull TransitionFactory<Drawable> drawableCrossFadeFactory) {
    return new BitmapTransitionOptions().transitionUsing(drawableCrossFadeFactory);
  }

  @NonNull
  public static BitmapTransitionOptions with(@NonNull TransitionFactory<Bitmap> transitionFactory) {
    return new BitmapTransitionOptions().transition(transitionFactory);
  }

  @NonNull
  public BitmapTransitionOptions crossFade() {
    return crossFade(new DrawableCrossFadeFactory.Builder());
  }

  @NonNull
  public BitmapTransitionOptions transitionUsing(
      @NonNull TransitionFactory<Drawable> drawableCrossFadeFactory) {
```

After

```
public class newClass{
  @NonNull
  public static BitmapTransitionOptions withWrapped(
      @NonNull TransitionFactory<Drawable> drawableCrossFadeFactory) {
    return new BitmapTransitionOptions().transitionUsing(drawableCrossFadeFactory);
  }

  @NonNull
  public static BitmapTransitionOptions with(@NonNull TransitionFactory<Bitmap> transitionFactory) {
    return new BitmapTransitionOptions().transition(transitionFactory);
  }

  @NonNull
  public BitmapTransitionOptions crossFade() {
    return crossFade(new DrawableCrossFadeFactory.Builder());
  }
}
```

1 2 3 4 5

Stro ◯ ◯ ◯ ◯ ◯ Strongly agree

Extract Variable

"Extract Variable" is a code refactoring technique used to improve code readability and maintainability by breaking down complex expressions or values into named variables with meaningful names.

1. The presented image illustrates an example of "Extract Variable".

   After applying the respective refactoring, the code has improved, now having higher quality, being less complex, and easier to understand, analyze, and modify.

   Before

   ```
   return getPhotoCacheDir(context, DEFAULT_DISK_CACHE_DIR);
   ```

   After

   ```
   File photoCacheDir = getPhotoCacheDir(context, DEFAULT_DISK_CACHE_DIR);
   return photoCacheDir;
   ```

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly agree

Move Method

"Move Method" is a code refactoring technique that involves relocating a method from one class to another to enhance code organization and improve maintainability.

1. The presented image illustrates an example of "Move Method".

   After applying the respective refactoring, the code has improved, now having higher quality, being less complex, and easier to understand, analyze, and modify.

Before

```
public class DevTools implements Closeable {
  private static final Logger LOG = Logger.getLogger(DevTools.class.getName());

  private final Domains protocol;
  private final Duration timeout = Duration.ofSeconds(10);
  private final Connection connection;
  private SessionID cdpSession = null;

  public DevTools(Function<DevTools, Domains> protocol, Connection connection) {
    this.connection = Require.nonNull( argName: "WebSocket connection", connection);
    this.protocol = Require.nonNull( argName: "CDP protocol", protocol).apply( t: this);
  }

  public Domains getDomains() { return protocol; }

  @Override
  public void close() {...}

  public void disconnectSession() {...}

  public <X> X send(Command<X> command) {...}

  public <X> void addListener(Event<X> event, Consumer<X> handler) {...}

  public void clearListeners() {...}
```

After

```
public class DevTools implements Closeable {
  private static final Logger LOG = Logger.getLogger(DevTools.class.getName());

  private final Domains protocol;
  private final Duration timeout = Duration.ofSeconds(10);
  private final Connection connection;
  private SessionID cdpSession = null;

  public DevTools(Function<DevTools, Domains> protocol, Connection connection) {
    this.connection = Require.nonNull( argName: "WebSocket connection", connection);
    this.protocol = Require.nonNull( argName: "CDP protocol", protocol).apply( t: this);
```

```
}

public Domains getDomains() { return protocol; }

@Override
public void close() {...}

public <X> X send(Command<X> command) {...}

public <X> void addListener(Event<X> event, Consumer<X> handler) {...}

public void clearListeners() {...}

public void createSessionIfThereIsNotOne() { createSessionIfThereIsNotOne( windowHandle: null); }
```

Target Class

```
public class Connection implements Closeable {

  private static final Logger LOG = Logger.getLogger(Connection.class.getName());
  private static final Json JSON = new Json();
  private static final Executor EXECUTOR =
      Executors.newCachedThreadPool(
          r -> {
            Thread thread = new Thread(r, name: "BiDi Connection");
            thread.setDaemon(true);
            return thread;
          });
  private static final AtomicLong NEXT_ID = new AtomicLong( initialValue: 1L);
  private final WebSocket socket;
  private final Map<Long, Consumer<Either<Throwable, JsonInput>>> methodCallbacks =
      new ConcurrentHashMap<>();
  private final ReadWriteLock callbacksLock = new ReentrantReadWriteLock( fair: true);
  private final HttpClient client;

  public Connection(HttpClient client, String url) {...}

  public void disconnectSession() {...}

  private static class NamedConsumer<> implements Consumer<X> {...}
```

1 2 3 4 5

Stro ◯ ◯ ◯ ◯ ◯ Strongly agree

Introduce Parameter Object

"Introduce Parameter Object" is a refactoring technique used in software development to improve cod by grouping related parameters into a single object or class.

1. The presented image illustrates an example of "Introduce Parameter Object".

   After applying the respective refactoring, the code has improved, now having higher quality, being less complex, and easier to understand, analyze, and modify.

Before

```
BufferedStartupStep(BufferedStartupStep parent, String name, long id, Instant startTime,
        Consumer<BufferedStartupStep> recorder) {
    this.parent = parent;
    this.name = name;
    this.id = id;
    this.startTime = startTime;
    this.recorder = recorder;
}
```

After

```
BufferedStartupStep(Startup startup) {
    this.parent = startup.getParent();
    this.name = startup.getName();
    this.id = startup.getId();
    this.startTime = startup.getStartTime();
    this.recorder = startup.getRecorder();
}
```

1 2 3 4 5

Stro ◯ ◯ ◯ ◯ ◯ Strongly agree

Replace Inheritance with Delegation

"Replace Inheritance with Delegation" is a refactoring technique that involves replacing inheritance relationships between classes with delegation, typically by creating an instance of another class and forwarding calls to it instead of inheriting behaviour directly.

1. The presented image illustrates an example of "Replace Inheritance with Delegation."

   After applying the respective refactoring, the code has improved, now having higher quality, being less complex, and easier to understand, analyze, and modify.

Before

```
public abstract class DeferredScalarObserver<T, R>
extends DeferredScalarDisposable<R>
implements Observer<T> {

    private static final long serialVersionUID = -2661951754089886511L;

    /** The upstream disposable. */
    protected Disposable upstream;

    /** Creates a DeferredScalarObserver instance and wraps a downstream Observer. ...*/
    public DeferredScalarObserver(Observer<? super R> downstream) { super(downstream); }

    @Override
    public void onSubscribe(Disposable d) {
        if (DisposableHelper.validate(this.upstream, d)) {...}
    }
```

After

```
public abstract class DeferredScalarObserver<T, R>
        implements Observer<T> {

    private static final long serialVersionUID = -2661951754089886511L;
    protected final MyDeferredScalarDisposable deferredScalarDisposable;

public void onError(Throwable t) {
    deferredScalarDisposable.value = null;
    error(t);
}

@Override
public void onComplete() {
    R v = deferredScalarDisposable.value;
    if (v != null) {
        deferredScalarDisposable.value = null;
        complete(v);
    } else {
        complete();
    }
}
```

1 2 3 4 5

Stro ◯ ◯ ◯ ◯ ◯ Strongly agree